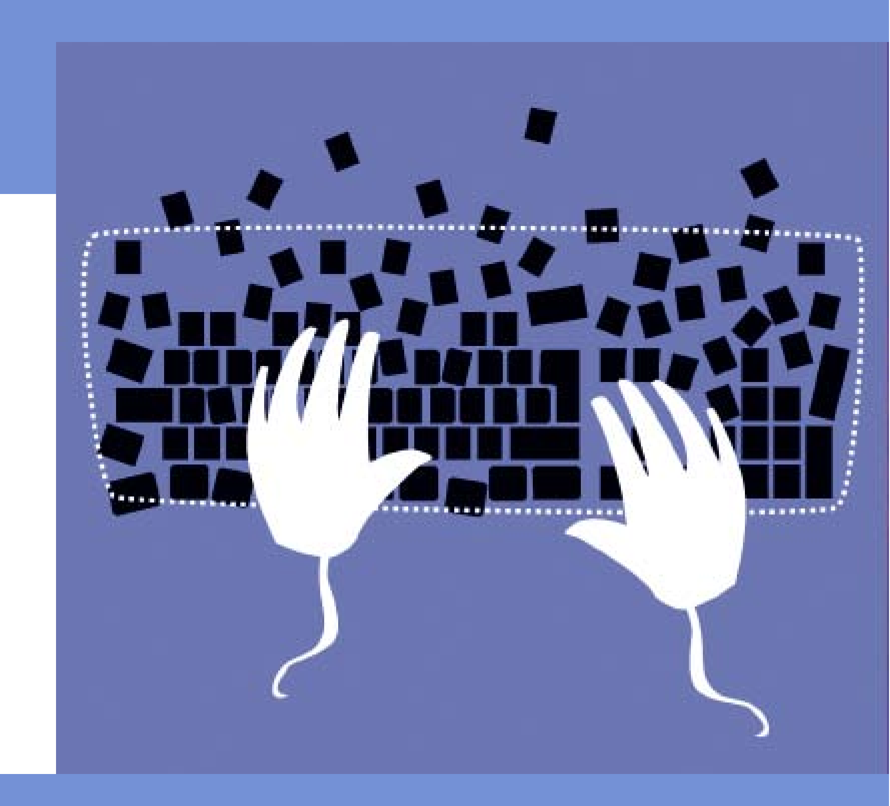
Software libre



Carles Mateu

XP04/90791/00021

Desarrollo de aplicaciones web



David Megías Jiménez

Coordinador

Ingeniero en Informática por la UAB.

Magíster en Técnicas Avanzadas de Automatización de Procesos por la UAB.

Doctor en Informática por la UAB.

Profesor de los Estudios de Informática y Multimedia de la UOC.

Jordi Mas

Coordinador

Ingeniero de software en la empresa de código abierto Ximian, donde trabaja en la implementación del proyecto libre Mono. Como voluntario, colabora en el desarrollo del procesador de textos Abiword y en la ingeniería de las versiones en catalán del proyecto Mozilla y Gnome. Es también coordinador general de Softcatalà. Como consultor ha trabajado para empresas como Menta, Telépolis, Vodafone, Lotus, eresMas, Amena y Terra España.

Carles Mateu

Autor

Ingeniero en Informatica por la UOC. Actualmente es Director del Área de Sistemas de Información y Comunicaciones de la UdL y profesor asociado de Redes e Internet en la UdL.

Primera edición: marzo 2004

Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-7.599-2004
ISBN: 84-9788-118-4

Índice

Agradecimientos

El autor agradece a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la primera edición de esta obra, enmarcada en el Máster Internacional en Software Libre ofrecido por la citada institución.

Asimismo, agradece el trabajo de Alberto Otero, cuya revisión ha fructificado en un manuscrito mejor. La corrección es mérito suyo, las incorrecciones demérito exclusivo del autor.

Y finalmente le da las gracias a Ruth Valencia por la imagen de la portada y por su amistad.

ANOTACIONES

1. Introducción a las aplicaciones web

1.1. Introducción a Internet

Internet, la red de redes, nace a mediados de la década de los setenta, bajo los auspicios de DARPA, la Agencia de Proyectos Avanzados para la Defensa de Estados Unidos. DARPA inició un programa de investigación de técnicas y tecnologías para unir diversas redes de conmutación de paquetes, permitiendo así a los ordenadores conectados a estas redes comunicarse entre sí de forma fácil y transparente.

De estos proyectos nació un protocolo de comunicaciones de datos, IP o Internet Protocol, que permitía a ordenadores diversos comunicarse a través de una red, Internet, formada por la interconexión de diversas redes.

A mediados de los ochenta la Fundación Nacional para la Ciencia norteamericana, la NSF, creó una red, la NSFNET, que se convirtió en el *backbone* (el troncal) de Internet junto con otras redes similares creadas por la NASA (NSINet) y el U.S. DoE (Department of Energy) con la ESNET. En Europa, la mayoría de países disponían de *backbones* nacionales (NORDUNET, RedIRIS, SWITCH, etc.) y de una serie de iniciativas paneuropeas (EARN y RARE). En esta época aparecen los primeros proveedores de acceso a Internet privados que ofrecen acceso pagado a Internet.

A partir de esta época, gracias entre otras cosas a la amplia disponibilidad de implementaciones de la *suite* de protocolos TCP/IP (formada por todos los protocolos de Internet y no sólo por TCP e IP), algunas de las cuales eran ya de código libre, Internet empezó lo que posteriormente se convertiría en una de sus características fundamentales, un ritmo de crecimiento exponencial, hasta que a mediados del 2002 empieza a descender ligeramente el ritmo de crecimiento.

ANOTACIONES

A mediados de los noventa se inició el *boom* de Internet. En esa época el número de proveedores de acceso privado se disparó, permitiendo a millones de personas acceder a Internet, que a partir de ese momento ya se empezó a conocer como la Red, desbancado a las demás redes de comunicación existentes (Compuserve, FidoNet/BBS, etc.). El punto de inflexión vino marcado por la aparición de implementaciones de TCP/IP gratuitas (incluso de implementaciones que formaban parte del sistema operativo) así como por la popularización y abaratamiento de medios de acceso cada vez más rápidos (módems de mayor velocidad, RDSI, ADSL, cable, satélite). El efecto de todos estos cambios fue de “bola de nieve”: a medida que se conectaban más usuarios, los costes se reducían, aparecían más proveedores e Internet se hacía más atractivo y económico, con lo que se conectaban más usuarios, etc.

En estos momentos disponer de una dirección de correo electrónico, de acceso a la web, etc., ha dejado de ser una novedad para convertirse en algo normal en muchos países del mundo. Por eso las empresas, instituciones, administraciones y demás están migrando rápidamente todos sus servicios, aplicaciones, tiendas, etc., a un entorno web que permita a sus clientes y usuarios acceder a todo ello por Internet. A pesar del ligero descenso experimentado en el ritmo de crecimiento, Internet está destinado a convertirse en una suerte de servicio universal de comunicaciones, permitiendo una comunicación universal.

1.2. La WWW como servicio de Internet

La WWW (World Wide Web) o, de forma más coloquial, la web, se ha convertido, junto con el correo electrónico, en el principal caballo de batalla de Internet. Ésta ha dejado de ser una inmensa “biblioteca” de páginas estáticas para convertirse en un servicio que permite acceder a multitud de prestaciones y funciones, así como a infinidad de servicios, programas, tiendas, etc.

1.2.1. Breve historia de la WWW

En 1989, mientras trabajaba en el CERN (Centro Europeo de Investigación Nuclear), Tim Berners-Lee empezó a diseñar un sistema

ANOTACIONES

para hacer accesible fácilmente la información del CERN. Dicho sistema empleaba el hipertexto para estructurar una red de enlaces entre los documentos. Una vez obtenida la aprobación para continuar el proyecto, nació el primer navegador web, llamado World-WideWeb (sin espacios).

En 1992 el sistema ya se había extendido fuera del CERN. El número de servidores "estables" había aumentado, alcanzando la sorprendente cifra de veintiséis. A partir de este punto, el crecimiento es espectacular. En 1993 la web ya era merecedora de un espacio en el *New York Times*.

Éste es el año del lanzamiento de Mosaic, un navegador para X-Window/Unís que con el tiempo se convertiría en Netscape y que fue un factor clave de popularización de la web. En 1994 se fundó el WWW Consortium, que se convertiría en el motor de desarrollo de los estándares predominantes en la web (http://www.w3c.org). A partir de ese momento, el crecimiento ya fue constante, convirtiéndose hacia finales de los noventa en el servicio insignia de Internet y dando lugar al crecimiento imparable de los servicios en línea que estamos experimentado actualmente.

1.2.2. Fundamentos de la web

El éxito espectacular de la web se basa en dos puntales fundamentales: el protocolo HTTP y el lenguaje HTML. Uno permite una implementación simple y sencilla de un sistema de comunicaciones que nos permite enviar cualquier tipo de ficheros de una forma fácil, simplificando el funcionamiento del servidor y permitiendo que servidores poco potentes atiendan miles de peticiones y reduzcan los costes de despliegue. El otro nos proporciona un mecanismo de composición de páginas enlazadas simple y fácil, altamente eficiente y de uso muy simple.

El protocolo HTTP

El protocolo HTTP (*hypertext tranfer protocol*) es el protocolo base de la WWW. Se trata de un protocolo simple, orientado a conexión y sin

ANOTACIONES

estado. La razón de que esté orientado a conexión es que emplea para su funcionamiento un protocolo de comunicaciones (TCP, *transport control protocol*) de modo conectado, un protocolo que establece un canal de comunicaciones de extremo a extremo (entre el cliente y el servidor) por el que pasa el flujo de bytes que constituyen los datos que hay que transferir, en contraposición a los protocolos de datagrama o no orientados a conexión que dividen los datos en pequeños paquetes (datagramas) y los envían, pudiendo llegar por vías diferentes del servidor al cliente. El protocolo no mantiene estado, es decir, cada transferencia de datos es una conexión independiente de la anterior, sin relación alguna entre ellas, hasta el punto de que para transferir una página web tenemos que enviar el código HTML del texto, así como las imágenes que la componen, pues en la especificación inicial de HTTP, la 1.0, se abrían y usaban tantas conexiones como componentes tenía la página, trasfiriéndose por cada conexión un componente (el texto de la página o cada una de las imágenes).

Nota

HTTP utiliza el puerto 80 (equivalente de alguna forma al identificador de conexión o de servicio TCP) para todas las conexiones por defecto (podemos utilizar otros puertos diferentes del 80).

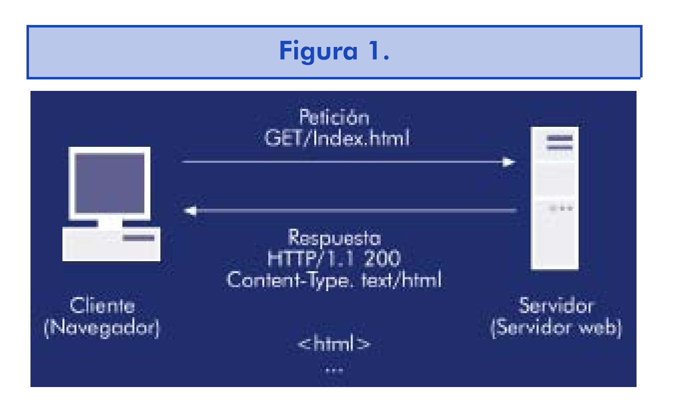
Existe una variante de HTTP llamada HTTPS (S por *secure*) que utiliza el protocolo de seguridad SSL (*secure socket layer*) para cifrar y autenticar el tráfico entre cliente y servidor, siendo ésta muy usada por los servidores web de comercio electrónico, así como por aquellos que contienen información personal o confidencial.

Nota

HTTPS utiliza por defecto el puerto 443.

De manera esquemática, el funcionamiento de HTTP es el siguiente: el cliente establece una conexión TCP hacia el servidor, hacia el puerto HTTP (o el indicado en la dirección de conexión), envía un comando HTTP de petición de un recurso (junto con algunas cabeceras informativas) y por la misma conexión el servidor responde con los datos solicitados y con algunas cabeceras informativas.

Figura 1.



ANOTACIONES

El protocolo define además cómo codificar el paso de parámetros entre páginas, el tunelizar las conexiones (para sistemas de *firewall*), define la existencia de servidores intermedios de *cache,* etc.

Las directivas de petición de información que define HTTP 1.1 (la versión considerada estable y al uso) son:

GET Petición de recurso.
POST Petición de recurso pasando parámetros.
HEAD Petición de datos sobre recurso.
PUT Creación o envío de recurso.
DELETE Eliminación de recurso.
TRACE Devuelve al origen la petición tal como se ha recibido en el receptor, para depurar errores.
OPTIONS Sirve para comprobar las capacidades del servidor.
CONNECT Reservado para uso en servidores intermedios capaces de funcionar como túneles.

Detallaremos a continuación algunos de estos comandos, ya que su comprensión es fundamental para el desarrollo de aplicaciones web.

Cabe destacar que todos los recursos que sean servidos mediante HTTP deberán ser referenciados mediante una URL (*universal resource locators*).

- **Peticiones en HTTP: GET y POST**

Las peticiones en HTTP pueden realizarse usando dos métodos. El método GET, en caso de enviar parámetros junto a la petición, las enviaría codificadas en la URL. Por su parte, el método POST, en caso de enviarlos, lo haría como parte del cuerpo de la petición.

Una petición GET sigue el siguiente formato:

```
GET /index.html HTTP/1.1
Host: www.ejemplo.com
User-Agent: Mozilla/4.5 [en]
Accept: image/gif, image/jpeg, text/html
Accept-language: en
Accept-Charset: iso-8859-1
```

ANOTACIONES

Podemos ver que está formada por:

1. **Línea de petición**: contiene el recurso solicitado.
2. **Cabecera de petición**: contiene información adicional sobre el cliente.
3. **Cuerpo de petición**: en las peticiones de tipo POST, y otras, contiene información adicional.

- **Línea de petición**

La línea de petición está formada por los siguientes elementos:

1. **Método**: nombre del método de HTTP llamado (GET, POST, etc.).
2. **Identificador de recurso**: URL (*uniform resource locator*) del recurso solicitado.
3. **Versión de protocolo**: versión del protocolo solicitada para la respuesta.

- **Cabecera de petición**

Contiene información adicional que puede ayudar al servidor (o a los servidores intermedios, los *proxies y caches*) a procesar adecuadamente la petición.

La información se proporciona en forma de:

Identificador: valor

De estos identificadores, los más conocidos e importantes son:

Host: nombre del servidor solicitado.
User-Agent: nombre del navegador o programa usado para acceder al recurso.
Accept: algunos formatos de texto e imagen aceptados por el cliente.
Accept-Language: idiomas soportados (preferidos) por el cliente, útil para personalizar la respuesta automáticamente.

ANOTACIONES

- Parámetros de petición

Una petición HTTP puede también contener parámetros, como respuesta, por ejemplo, a un formulario de registro, a una selección de producto en una tienda electrónica, etc. Estos parámetros pueden pasarse de dos formas:

– Como parte de la cadena de petición, codificados como parte de la URL.

– Como datos extra a la petición.

Para codificar los parámetros como parte de la URL, éstos se añaden a la URL detrás del nombre del recurso, separados de éste por un carácter **?**. Los diferentes parámetros se separan entre sí por el carácter **&**. Los espacios se sustituyen por **+**. Por último, los caracteres especiales (los mencionados antes de **&**, **+** y **?**, así como los caracteres no imprimibles, etc.) se representan con **%xx**, donde **xx** representa al código ASCII en hexadecimal del carácter.

Por ejemplo:

```
http://www.ejemplo.com/indice.jsp?nombre=Perico+Palotes&OK=1
```

que en la petición HTTP quedaría:

```
GET /indice.jsp?nombre=Perico+Palotes&OK=1 HTTP/1.0
Host: www.ejemplo.com
User-Agent: Mozilla/4.5 [en]
Accept: image/gif, image/jpeg, text/html
Accept-language: en
Accept-Charset: iso-8859-1
```

Para pasar los parámetros como datos extra de la petición, éstos se envían al servidor como cuerpo de mensaje en la petición.

Por ejemplo, la petición anterior quedaría:

```
POST /indice.jsp HTTP/1.0
Host: www.ejemplo.com
```

ANOTACIONES

```
User-Agent: Mozilla/4.5 [en]
Accept: image/gif, image/jpeg, text/html
Accept-language: en
Accept-Charset: iso-8859-1

nombre=Perico+Palotes&OK=1
```

Cabe destacar que para pasar los parámetros como cuerpo de la petición, ésta debe realizarse como POST y no como GET, aunque una petición POST también puede llevar parámetros en la línea de petición. Los parámetros pasados como cuerpo de la petición están codificados, al igual que en el ejemplo anterior, como URL, o pueden usar una codificación derivada del formato MIME (*multipurpose internet mail extensions*), en lo que se conoce como codificación multiparte.

La petición anterior en formato multiparte sería:

```
POST /indice.jsp HTTP/1.0
Host: www.ejemplo.com
User-Agent: Mozilla/4.5 [en]
Accept: image/gif, image/jpeg, text/html
Accept-language: en
Accept-Charset: iso-8859-1
Content-Type: multipart/form-data,
      delimiter="----ALEATORIO----"

----ALEATORIO----
Content-Disposition: form-data; name="nombre"
Perico Palotes
----ALEATORIO----
Content-Disposition: form-data; name="OK"
1

----ALEATORIO------
```

Esta codificación es exclusiva del método POST. Se emplea para enviar ficheros al servidor.

- **Respuestas en HTTP**

Las respuestas en HTTP son muy similares a las peticiones. Una respuesta *estándar* a una petición de una página sería similar a lo siguiente:

ANOTACIONES

```
HTTP/1.1 200 OK
Date: Mon, 04 Aug 2003 15:19:10 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Tue, 25 Mar 2003 08:52:53 GMT
Accept-Ranges: bytes
Content-Length: 428
Connection: close

<HTML>
...
```

En ella podemos observar que la primera línea nos responde con la versión del protocolo empleada para enviarnos la página, seguida de un código de retorno y una frase de retorno. El código de retorno puede adoptar uno de los siguientes valores:

– 1xx Petición recibida, continúa en proceso.

– 2xx Correcta. Petición procesada correctamente.

– 3xx Redirección. La petición debe repetirse o redirigirse.

– 4xx Error de cliente. No se puede procesar la petición porque ésta es incorrecta, no existe, etc.

– 5xx Error de servidor. El servidor ha fallado intentando procesar la petición, que *a priori* es correcta.

La frase de retorno dependerá de la implementación, pero sólo sirve como aclaratorio del código de retorno.

Después del estatus aparece una serie de campos de control, con el mismo formato que en las cabeceras de la petición que nos informan del contenido (fecha de creación, longitud, versión del servidor, etc.). A continuación viene el contenido solicitado.

El lenguaje HTML

El otro puntal del éxito del WWW ha sido el lenguaje HTML (*hypertext mark-up language*). Se trata de un lenguaje de marcas (se utiliza insertando marcas en el interior del texto) que nos permite represen-

ANOTACIONES

tar de forma rica el contenido y también referenciar otros recursos (imágenes, etc.), enlaces a otros documentos (la característica más destacada del WWW), mostrar formularios para posteriormente procesarlos, etc.

El lenguaje HTML actualmente se encuentra en la versión 4.01 y empieza a proporcionar funcionalidades más avanzadas para crear páginas más ricas en contenido. Además se ha definido una especificación compatible con HTML, el XHTML (*extensible hypertext markup language*) que se suele definir como una versión XML validable de HTML, proporcionándonos un XML Schema contra el que validar el documento para comprobar si está bien formado, etc.

1.3. Historia de las aplicaciones web

Inicialmente la web era simplemente una colección de páginas estáticas, documentos, etc., que podían consultarse o descargarse.

El siguiente paso en su evolución fue la inclusión de un método para confeccionar páginas dinámicas que permitiesen que lo mostrado fuese dinámico (generado o calculado a partir de los datos de la petición). Dicho método fue conocido como CGI (*common gateway interface*) y definía un mecanismo mediante el cual podíamos pasar información entre el servidor HTTP y programas externos. Los CGI siguen siendo muy utilizados, puesto que la mayoría de los servidores web los soportan debido a su sencillez. Además, nos proporcionan total libertad a la hora de escoger el lenguaje de programación para desarrollarlos.

El esquema de funcionamiento de los CGI tenía un punto débil: cada vez que recibíamos una petición, el servidor web lanzaba un proceso que ejecutaba el programa CGI. Como, por otro lado, la mayoría de CGI estaban escritos en algún lenguaje interpretado (Perl, Python, etc.) o en algún lenguaje que requería *run-time environment* (VisualBasic, Java, etc.), esto implicaba una gran carga para la máquina del servidor. Además, si la web tenía muchos accesos al CGI, esto suponía problemas graves.

ANOTACIONES

Por ello se empiezan a desarrollar alternativas a los CGI para solucionar este grave problema de rendimiento. Las soluciones vienen principalmente por dos vías. Por un lado se diseñan sistemas de ejecución de módulos más integrados con el servidor, que evitan que éste tenga que instanciar y ejecutar multitud de programas. La otra vía consiste en dotar al servidor de un intérprete de algún lenguaje de programación (RXML, PHP, VBScript, etc.) que nos permita incluir las páginas en el código de manera que el servidor sea quien lo ejecute, reduciendo así el tiempo de respuesta.

A partir de este momento, se vive una explosión del número de arquitecturas y lenguajes de programación que nos permiten desarrollar aplicaciones web. Todas ellas siguen alguna de las dos vías ya mencionadas. De ellas, las más útiles y las que más se utilizan son aquellas que permiten mezclar los dos sistemas, es decir, un lenguaje de programación integrado que permita al servidor interpretar comandos que "incrustemos" en las páginas HTML y un sistema de ejecución de programas más enlazado con el servidor que no presente los problemas de rendimiento de los CGI.

A lo largo del curso abordaremos con más detalle la que quizás sea la más exitosa y potente de estas aproximaciones, la seguida por Sun Microsystems con su sistema Java, que está integrada por dos componentes; a saber, un lenguaje que permite incrustar código interpretable en las páginas HTML y que el servidor traduce a programas ejecutables, JSP (*Java server pages*) y un mecanismo de programación estrechamente ligado al servidor, con un rendimiento muy superior a los CGI convencionales, llamado Java Servlet.

Otra de las tecnologías que más éxito ha obtenido y una de las que más se utiliza en Internet es el lenguaje de programación interpretado por el servidor PHP. Se trata de un lenguaje que permite incrustar HTML en los programas, con una sintaxis que proviene de C y Perl. Además, habida cuenta de su facilidad de aprendizaje, su sencillez y potencia, se está convirtiendo en una herramienta muy utilizada para algunos desarrollos.

Otros métodos de programación de aplicaciones web también tienen su mercado. Así sucede con `mod_perl` para Apache, RXML para Roxen, etc., pero muchos de ellos están vinculados a un servidor web concreto.

ANOTACIONES

2. Instalación del servidor

2.1. Conceptos básicos del servidor web

Un servidor web es un programa que atiende y responde a las diversas peticiones de los navegadores, proporcionándoles los recursos que solicitan mediante el protocolo HTTP o el protocolo HTTPS (la versión segura, cifrada y autenticada de HTTP). Un servidor web básico tiene un esquema de funcionamiento muy sencillo, ejecutando de forma infinita el bucle siguiente:

1. Espera peticiones en el puerto TCP asignado (el estándar para HTTP es el 80).
2. Recibe una petición.
3. Busca el recurso en la cadena de petición.
4. Envía el recurso por la misma conexión por donde ha recibido la petición.
5. Vuelve al punto 2.

Un servidor web que siguiese el esquema anterior cumpliría los requisitos básicos de los servidores HTTP, aunque, eso sí, sólo podría servir ficheros estáticos.

A partir del esquema anterior se han diseñado y construido todos los programas servidores de HTTP que existen, variando sólo el tipo de peticiones (páginas estáticas, CGI, Servlets, etc.) que pueden atender, en función de que sean o no multi-proceso, multi-hilados, etc. A continuación detallaremos algunas de las características principales de los servidores web, que extienden, obviamente el esquema anterior.

2.1.1. Servicio de ficheros estáticos

Todos los servidores web deben incluir, como mínimo, la capacidad para servir los ficheros estáticos que se encuentren en alguna parte

ANOTACIONES

concreta del disco. Un requisito imprescindible es la capacidad de especificar qué parte del disco se servirá. No resulta en absoluto recomendable que el servidor nos obligue a usar un directorio concreto, si bien puede tener uno por defecto.

La mayoría de servidores web permiten, además, añadir otros directorios para servir, especificando en qué punto del "sistema de ficheros" virtual del servidor se ubicarán.

Ejemplo

Por ejemplo, podemos tener la siguiente situación:

Directorio del disco	Directorio web
/home/apache/html	/
/home/empresa/docs	/docs
/home/jose/informe	/informe-2003

En este caso, el servidor debería traducir las siguientes direcciones web de la siguiente manera:

URL	Fichero de disco
/index.html	/home/apache/html/index.html
/docs/manuales/producto.pdf	/home/empresa/docs/manuales/producto.pdf
/empresa/quienes.html	/home/apache/html/empresa/quienes.html
/informe-2003/index.html	/home/jose/informe/index.html

Algunos servidores web permiten, además, especificar directivas de seguridad (para qué direcciones, usuarios, etc., está visible un directorio, etc.), mientras que otros hacen posible especificar qué ficheros se considerarán como índice del directorio.

2.1.2. Seguridad y autenticación

La mayoría de los servidores web modernos nos permiten controlar desde el programa servidor aquellos aspectos relacionados con la seguridad y la autenticación de los usuarios.

ANOTACIONES

El modo más simple de control es el proporcionado por el uso de ficheros .htaccess. Éste es un sistema de seguridad que proviene de uno de los primeros servidores web (del NCSA httpd), que consiste en poner un fichero de nombre .htaccess en cualquier directorio del contenido web que se vaya a servir, indicando en este fichero qué usuarios, máquinas, etc., tienen acceso a los ficheros y subdirectorios del directorio donde está el fichero. Como el servidor de NCSA fue el servidor más usado durante mucho tiempo, la mayoría de servidores modernos permiten utilizar el fichero .htaccess respetando la sintaxis del servidor de NCSA.

Otros servidores permiten especificar reglas de servicio de directorios y ficheros en la configuración del servidor web, indicando allí qué usuarios, máquinas, etc., pueden acceder al recurso indicado.

Por lo que respecta a la autenticación (validación del nombre de usuario y contraseña proporcionados por el cliente), las prestaciones ofrecidas por los diversos servidores web son de lo más variado. La mayoría permiten, como mínimo, proporcionar al servidor web un fichero con nombres de usuario y contraseñas contra el que se pueda validar lo enviado por el cliente. De todos modos, es frecuente que los servidores proporcionen pasarelas que permitan delegar las tareas de autenticación y validación a otro software (por ejemplo RADIUS, LDAP, etc.). Si usamos un sistema operativo como Linux, que dispone de una infraestructura de autenticación como PAM (*pluggable authentication modules*), podemos usar esta funcionalidad como modo de autenticación del servidor web, permitiéndonos así usar los múltiples métodos disponibles en PAM para autenticar contra diversos sistemas de seguridad.

2.1.3. Contenido dinámico

Uno de los aspectos más importantes del servidor web escogido es el nivel de soporte que nos ofrece para servir contenido dinámico. Dado que la mayor parte del contenido web que se sirve no proviene de páginas estáticas, sino que se genera dinámicamente, y esta tendencia es claramente alcista, el soporte para contenido dinámico que nos ofrece el servidor web es uno de los puntos más críticos en su elección.

ANOTACIONES

La mayoría de servidores web ofrecen soporte para CGI (cabe recordar que los CGI son el método más antiguo y simple de generación de contenido dinámico). Muchos ofrecen soporte para algunos lenguajes de programación (básicamente interpretados) como PHP, JSP, ASP, Pike, etc. Es altamente recomendable que el servidor web que utilicemos proporcione soporte para alguno de estos lenguajes, siendo uno de los más utilizados PHP, sin tener en cuenta JSP, que usualmente requiere un software externo al servidor web para funcionar (como por ejemplo, un contenedor de Servlets). La oferta en este campo es muy amplia, pero antes de escoger un lenguaje de programación de servidor tenemos que plantearnos si deseamos un lenguaje muy estandarizado para que nuestra aplicación no dependa de un servidor web o arquitectura concreta o si, por el contrario, la portabilidad no es una prioridad y sí lo es alguna prestación concreta que pueda ofrecernos algún lenguaje de programación concreto.

2.1.4. Servidores virtuales

Una prestación que está ganado adeptos y usuarios a marchas forzadas, especialmente entre los proveedores de servicios de Internet y los operadores de alojamiento de dominios, es la capacidad de algunos servidores web de proporcionar múltiples dominios con sólo una dirección IP, discriminando entre los diversos dominios alojados por el nombre de dominio enviado en la cabecera de la petición HTTP. Esta prestación permite administrar de una forma más racional y ahorrativa un bien escaso, como son las direcciones IP.

Si necesitamos disponer de muchos nombres de servidor (ya sea porque proporcionamos alojamiento o por otros motivos) debemos asegurarnos de que el servidor web escogido ofrezca estas facilidades, y además, que el soporte que facilita para servidores virtuales nos permita una configuración diferente para cada servidor (directorios, usuarios, seguridad, etc.). Lo ideal sería que cada servidor se comportase como si de un ordenador diferente se tratase.

2.1.5. Prestaciones extra

Son muchas las prestaciones que nos ofrecen los servidores web para diferenciarse de la competencia. Algunas de ellas resultan realmente

ANOTACIONES

útiles y pueden decidir la elección de servidor web. Debemos ser conscientes, no obstante, de que si usamos algunas de estas características, o si éstas se convierten en imprescindibles para nosotros, ello nos puede obligar a usar un determinado servidor web aunque en algún momento determinado deseemos cambiar.

Algunas características adicionales de algunos servidores web de código libre son las siguientes:

Spelling (Apache). Esta prestación de Apache nos permite definir una página de error para los recursos no encontrados que sugiera al usuario algunos nombres de recurso parecidos al que solicitaba para el caso de que hubiese cometido un error al escribir.

Status (Apache). Nos proporciona una página web generada por el servidor donde éste nos muestra su estado de funcionamiento, nivel de respuesta, etc.

RXML Tags (Roxen). Añade a HTML algunos *tags* (comandos de HTML), mejorados para programación y generación de contenido dinámico.

SQL Tags (Roxen). Añade al lenguaje HTML extendido de Roxen (RXML), comandos para realizar accesos a bases de datos SQL desde las propias páginas HTML.

Graphics (Roxen). Añade al lenguaje HTML extendido de Roxen (RXML), comandos para la generación de gráficos, títulos, etc., y no requiere un trabajo de diseño gráfico.

Bfnsgd (AOLServer), **mod_gd** (Apache). Nos permite realizar gráficos a partir de texto y de fuentes *True Type*.

mod_mp3 (Apache), **ICECAST**, **MPEG** (Roxen). Nos permiten convertir el servidor web en un servidor de música (con *streaming*, etc.).

Throttle (Roxen), **mod_throttle** (Apache). Nos proporcionan medios para limitar la velocidad de servicio de HTTP, ya sea por usuario, servidor virtual, etc.

ANOTACIONES

Nsxml (AOLServer), **tDOM** (AOLServer), **mod_xslt** (Apache). Nos permite realizar la transformación de ficheros XML a partir de XSL.

Kill Frame (Roxen). Envía con cada página web servida un código que evita que nuestra web quede como *frame* (enmarcada) dentro de otra página web.

2.1.6. Actuación como representantes

Algunos servidores web nos permiten usarlos como servidores intermedios *(proxy servers*). Podemos usar los servidores intermedios para propósitos muy variados:

- Para servir de aceleradores de navegación de nuestros usuarios (uso como *proxy-cache*).
- Para servir como aceleradores de acceso frontales para un servidor web, instalando diversos servidores web que repliquen los accesos a un servidor maestro *(reverse-proxy* o *HTTP server acceleration*).
- Como frontales a algún servidor o protocolo.

Algunos servidores web nos permiten usarlos como servidores intermediarios para alguno de los usos anteriores. No obstante, para los dos primeros usos (aceleradores de navegación o de acceso) existen programas específicos de código libre mucho más eficientes, entre los que destaca Squid (http://www.squid-cache.org/), que se considera uno de los mejores productos de *proxy* existentes.

Existen módulos para diversos servidores web que nos permiten usarlos como frontales para otros servidores especializados en otro tipo de servicio.

Ejemplo

Por ejemplo, Tomcat es un motor de ejecución de Servlets y JSP, que incluye un pequeño servidor de HTTP para atender las peticiones de contenido estático y redirigir el

ANOTACIONES

resto al motor de ejecución de Servlets (los Servlets y los JSP son mecanismos de desarrollo de aplicaciones web), pero a pesar de incluir un servidor web, Apache es el servidor web usado por excelencia con Tomcat. Para ello disponemos de un módulo de Apache que realiza el enlace con Tomcat. Dicho módulo se denomina **mod_jk2**.

2.1.7. Protocolos adicionales

Algunos servidores, además de atender y servir peticiones HTTP (y HTTPS), pueden atender y servir peticiones de otros protocolos o de protocolos implementados sobre HTTP. Algunos de estos protocolos pueden convertirse en requisitos fundamentales de nuestro sistema y, por ello, su existencia en el del servidor web puede ser imprescindible.

2.2. El servidor Apache

Apache es un servidor web de código libre robusto cuya implementación se realiza de forma colaborativa, con prestaciones y funcionalidades equivalentes a las de los servidores comerciales. El proyecto está dirigido y controlado por un grupo de voluntarios de todo el mundo que, usando Internet y la web para comunicarse, planifican y desarrollan el servidor y la documentación relacionada.

Estos voluntarios se conocen como el *Apache Group*. Además del *Apache Group*, cientos de personas han contribuido al proyecto con código, ideas y documentación.

2.2.1. El nacimiento de Apache

En febrero de 1995, el servidor web más popular de Internet era un servidor de dominio público desarrollado en el NCSA (National Center for Supercomputing Applications en la Universidad de Illinois). No obstante, al dejar Rob McCool (el principal desarrollador del servidor) la

ANOTACIONES

NCSA en 1994, la evolución de dicho programa había quedado prácticamente reducida a la nada. El desarrollo pasó a manos de los responsables de sitios web, que progresivamente introdujeron mejoras en sus servidores. Un grupo de éstos, usando el correo electrónico como herramienta básica de coordinación, se puso de acuerdo en poner en común estas mejoras (en forma de "parches", *patches*). Dos de estos desarrolladores, Brian Behlendorf y Cliff Skolnick, pusieron en marcha una lista de correo, un espacio de información compartida y un servidor en California donde los desarrolladores principales pudiesen trabajar. A principios del año siguiente, ocho programadores formaron lo que sería el Grupo Apache.

Éstos, usando el servidor NCSA 1.3 como base de trabajo, añadieron todas las correcciones de errores publicadas y las mejoras más valiosas que encontraron y probaron el resultado en sus propios servidores. Posteriormente publicaron lo que sería la primera versión oficial del servidor Apache (la 0.6.2, de Abril de 1995). Casualmente, en esas mismas fechas, NCSA reemprendió el desarrollo del servidor NCSA.

En este momento el desarrollo de Apache siguió dos líneas paralelas. Por un lado, un grupo de los desarrolladores siguió trabajando sobre el Apache 0.6.2 para producir la serie 0.7, incorporando mejoras, etc. Un segundo grupo reescribió por completo el código, creando una nueva arquitectura modular. En julio de 1995 se migraron a esta nueva arquitectura las mejoras existentes para Apache 0.7, haciéndose público como Apache 0.8.

El día uno de diciembre de 1995, apareció Apache 1.0, que incluía documentación y muchas mejoras en forma de módulos incrustables. Poco después, Apache sobrepasó al servidor de NCSA como el más usado en Internet, posición que ha mantenido hasta nuestros días. En 1999 los miembros del Grupo Apache fundaron la Apache Software Foundation, que provee soporte legal y financiero al desarrollo del servidor Apache y los proyectos laterales que han surgido de éste.

2.2.2. Instalación de Apache

Tenemos dos opciones principales para instalar Apache: podemos compilar el código fuente o podemos instalarlo a partir de un paquete binario adecuado a nuestro sistema operativo.

ANOTACIONES

Compilación a partir de fuentes

Para compilar Apache a partir de código fuente, debemos obtener previamente de la web de Apache la última versión (http://httpd.apache.org) de éste. Una vez descargado, seguiremos los siguientes pasos:

Descomprimiremos el fichero que acabamos de descargar, lo cual nos creará un directorio donde estarán las fuentes del servidor.

Una vez dentro de este directorio, continuaremos con los siguientes pasos:

– Configuraremos el código para su compilación. Para ello ejecutaremos:

```
$ ./configure
```

Disponemos de algunos parámetros para ajustar la compilación de Apache.

Los más importantes son:

Tabla 1. Parámetros

Parámetro	Significado
`--prefix`	Directorio donde queremos instalar Apache
`--enable-modules =LISTA-MODULOS`	Módulos que hay que activar
`--enable-mods-shared =LISTA-MODULOS`	Módulos *shared* que hay que activar
`--enable-cache`	*Cache* dinámico
`--enable-disk-cache`	*Cache* dinámico en disco
`--enable-mem-cache`	Módulo de *cache* de memoria
`--enable-mime-magic`	Determinación tipo MIME automática
`--enable-usertrack`	Seguimiento de sesión de usuario
`--enable-proxy`	Módulo Apache-*proxy*
`--enable-proxy-connect`	Módulo Apache-*proxy* para CONNECT

ANOTACIONES

Parámetro	Significado
`--enable-proxy-ftp`	Módulo Apache-*proxy* para FTP
`--enable-proxy-http`	Módulo Apache-*proxy* HTTP
`--enable-ssl`	Soporte de SSL/TLS (mod ssl)
`--enable-http`	Manejo del protocolo HTTP
`--enable-dav`	Manejo del protocolo WebDAV
`--disable-cgid`	Soporte de CGI optimizado
`--enable-cgi`	Soporte de CGI
`--disable-cgi`	Soporte de CGI
`--enable-cgid`	Soporte de CGI optimizado
`--enable-vhost-alias`	Soporte de *hosts* virtuales

– Una vez configurado el código fuente, si no se han producido errores procederemos a compilarlo. Para ello ejecutaremos:

```
$ make
```

El alumno debe recordar que para compilar Apache requeriremos, como mínimo, GNU Make y GNU CC.

– Una vez compilado, podemos instalarlo en el directorio que hemos designado como destino en la configuración anterior, con `configure`. Este paso se realiza usando uno de los objetivos que ya tiene definidos `make`. Concretamente lo realizaremos con:

```
$ make install
```

– Una vez instalado en su lugar, dispondremos, dentro del subdirectorio bin del directorio de instalación, el que hemos especificado con `prefix`, un programa llamado `apachectl` que nos permitirá controlar el servidor. Para iniciar éste:

```
$cd <directorio de instalacion>/bin
$ ./apachectl start
```

Para detenerlo:

```
$cd <directorio de instalacion>/bin
$ ./apachectl stop
```

Instalación a partir de paquetes binarios

La mayor parte de los sistemas operativos de código libre, y especialmente la mayor parte de distribuciones de Linux, incluyen el servidor Apache. A pesar de ello, en muchos casos es necesario instalar Apache. Quizás no lo instalásemos en su momento. Así pues, ahora necesitamos una nueva versión. También es posible que deseemos reinstalarlo porque hemos tenido problemas con algún fichero.

A continuación ofreceremos unas cuantas indicaciones sobre la instalación de Apache en algunas de las distribuciones de Linux más conocidas.

- **Redhat/Fedora**

Las distribuciones de Redhat y Fedora incluyen desde hace tiempo el servidor Apache. En caso de tener que instalarlo el proceso es realmente simple.

Debemos descargarnos del servidor correspondiente (bien de redhat.com o de fedora.us) el paquete binario de Apache (en formato RPM). Tenemos que asegurarnos de que estamos descargando la última versión para nuestra distribución, ya que tanto Redhat como Fedora publican actualizaciones para subsanar errores o problemas detectados. Una vez en posesión de dicho paquete, debemos instalarlo con:

```
rpm -ihv httpd-x.x.x.rpm
```

En caso de que lo tuviésemos ya instalado, podremos actualizarlo mediante el comando:

```
rpm -Uhv httpd-x.x.x.rpm
```

En el caso de Fedora, al usar esta distribución un repositorio `apt` podemos actualizar o instalar Apache con:

```
apt-get install httpd
```

ANOTACIONES

Asimismo debemos instalar los módulos adicionales que deseemos, como por ejemplo:

Ejemplo

- mod_auth_*
- mod_python
- mod_jk2.
- mod_perl
- mod_ssl
- php
- etc.

- **Debian**

La instalación de Apache en Debian es realmente sencilla. Sólo tenemos que ejecutar el siguiente comando:

```
apt-get install apache
```

que nos instalará Apache en la última versión o bien lo actualizará, si ya lo teníamos instalado.

2.2.3. Configuración de Apache

Una vez instalado el servidor, llega el momento de configurarlo. Por defecto Apache incluye una configuración mínima que arranca el servidor en el puerto TCP de servicio por defecto (el puerto 80) y sirve todos los ficheros del directorio especificado por la directiva de configuración `DocumentRoot`. El fichero de configuración de Apache se llama `httpd.conf` y se encuentra en el subdirectorio `conf` del directorio de instalación. El fichero `httpd.conf` es un fichero ASCII con las directivas de configuración de Apache.

Estructura del fichero de configuración

El fichero `httpd.conf` está dividido en tres bloques fundamentales, aunque las directivas de cada bloque pueden aparecer mezcladas y desordenadas.

ANOTACIONES

Dichos bloques son:

- Parámetros globales
- Directivas de funcionamiento
- *Hosts* virtuales

Algunos parámetros son generales para el servidor, mientras que otros se pueden configurar de manera independiente para cada conjunto de directorios o ficheros o para un servidor virtual concreto. En estos casos, los parámetros se encuentran ubicados dentro de secciones donde se indica el ámbito de aplicación del parámetro.

Las secciones más importantes son:

`<Directory>`: los parámetros que se encuentran dentro de esta sección sólo se aplicarán al directorio especificado y a sus subdirectorios.

`<DirectoryMatch>`: igual que Directory, pero acepta en el nombre del directorio expresiones regulares.

`<Files>`: los parámetros de configuración proporcionan control de acceso de los ficheros por su nombre.

`<FilesMatch>`: igual que Files, pero acepta expresiones regulares en el nombre del fichero.

`<Location>`: proporciona un control de acceso de los ficheros por medio de la URL.

`<LocationMatch>`: igual que Location, pero acepta expresiones regulares en el nombre del fichero.

`<VirtualHost>`: los parámetros sólo se aplicarán a las peticiones que vayan dirigidas a este *host* (nombre de servidor, o dirección IP, o puerto TCP).

`<Proxy>`: sólo se aplican los parámetros a las peticiones de *proxy* (requiere, por tanto, *mod proxy* instalado) que coincidan con la especificación de URL.

ANOTACIONES

<ProxyMatch>: igual que *proxy*, pero acepta expresiones regulares en la URL especificada.

<IfDefine>: se aplica si al arrancar el servidor se define en la línea de comandos un parámetro concreto, con la opción -D.

<IfModule>: se aplican los parámetros si al arrancar el servidor el módulo especificado se encuentra cargado (con `LoadModule`).

En caso de conflicto entre especificaciones de parámetros, el orden de precedencia es el siguiente:

1. `<Directory>` y `.htaccess`
2. `<DirectoryMatch>` y `<Directory>`
3. `<Files>` y `<FilesMatch>`
4. `<Location>` y `<LocationMatch>`

Por lo que se refiere a `<VirtualHost>`, estas directivas siempre se aplican después de las generales. Así, un `VirtualHost` siempre puede sobrescribir la configuración por defecto.

Un ejemplo de configuración sería:

```
<Directory /home/*/public_html>
   Options Indexes
</Directory>
<FilesMatch \.(?i:gif jpe?g png)$>
   Order allow,deny
   Deny from all
</FilesMatch>.
```

Directivas de configuración globales

Algunas de las directivas de configuración jamás se aplican a ninguna de las secciones antes mencionadas (directorios, etc.), sino que afectan a todo el servidor web. Las más destacadas son:

- **`ServerRoot`**: especifica la ubicación del directorio raíz donde se encuentra instalado el servidor web. A partir de este direc-

ANOTACIONES

torio, encontraremos los ficheros de configuración, etc. Si el servidor está correctamente instalado, no debería cambiarse nunca.

- **`KeepAlive`**: especifica si se utilizarán conexiones persistentes, para atender todas las peticiones de un usuario con la misma conexión TCP.

- **`Listen`**: especifica en qué puerto se atenderán las peticiones. Por defecto se utiliza el puerto 80 de TCP. También permite especificar qué direcciones IP se utilizarán (por si el servidor tuviese más de una). Por defecto se utilizan todas las disponibles.

- **`LoadModule`**: con `LoadModule` podemos cargar en el servidor los módulos adicionales que incorpora Apache. La sintaxis es:

  ```
  LoadModule modulo ficheromodulo
  ```

  Debemos tener instalado `mod_so` para poder usarla.

Directivas principales

Disponemos de algunas directivas que, por norma general, no suelen estar en ninguna de las secciones anteriormente mencionadas (algunas de ellas no pueden estar dentro de ninguna sección y es obligatorio que estén en la principal), sino que se hallan en la sección principal de configuración. Dichas directivas son:

- **`ServerAdmin`**: sirve para especificar la dirección de correo electrónico del administrador. Esta dirección puede aparecer en los mensajes de error como dirección de contacto para permitir a los usuarios notificar un error al administrador. No puede estar dentro de ninguna sección.

- **`ServerName`**: especifica el nombre y el puerto TCP que el servidor utiliza para identificarse. Puede determinarse automáticamente, pero es recomendable especificarlo. Si el servidor no tiene un nombre DNS, se recomienda incluir su dirección IP. No puede estar dentro de ninguna

ANOTACIONES

sección. La sintaxis es: `ServerName nombredireccion:puerto` como en:

```
ServerName www.uoc.edu:80
ServerName 192.168.1.1:80
```

- **`DocumentRoot`**: el directorio raíz desde el que se servirán los documentos. Por defecto es el directorio `htdocs`, que se encuentra dentro de la carpeta de instalación de Apache. No puede estar dentro de ninguna sección, a excepción de la `VirtualHost`. Le corresponde una sección `<Directory>` en la que se establecen los parámetros de configuración de este directorio.

- **`DirectoryIndex`**: especifica el fichero que se servirá por defecto para cada directorio en el caso de que no se especifique ninguno en la URL. Por defecto es index.html. Es decir, si por ejemplo se pone en el navegador: www.uoc.edu el servidor por defecto enviará www.uoc.edu/index.html. Se puede especificar más de un fichero y el orden con el que se especifica dicho nombre determinará la prioridad para decidir cuál se sirve. La directiva se puede encontrar tanto fuera de cualquier sección como dentro de una de ellas.

- **`AccessFileName`**: especifica el nombre de fichero de configuración en caso de que éste sea diferente de .htaccess. Para que esta configuración funcione, la directiva AllowOverride tiene que tener un valor adecuado. No puede estar dentro de ninguna sección. El nombre de fichero que se especifica por defecto es el del fichero `.htaccess`.

- **`ErrorDocument`**: esta directiva establece la configuración del servidor en caso de error. Se pueden establecer cuatro configuraciones distintas:

  – Mostrar un texto de error.
  – Redirigir a un fichero en el mismo directorio.
  – Redirigir a un fichero en nuestro servidor.
  – Redirigir a un fichero fuera de nuestro servidor.

  La sintaxis de la directiva es `ErrorDocument` *`código_error acción`*.

ANOTACIONES

Esta directiva se puede encontrar tanto dentro de una sección, como en la configuración global.

Por ejemplo, con:

```
ErrorDocument 404 /noencont.html.
```

En caso de no encontrarse un fichero, se mostrará el fichero noencont.html.

- **`Alias`**: las directivas `Alias` y `AliasMatch` nos permiten definir accesos a directorios que se encuentran fuera del `DocumentRoot`. La sintaxis es la siguiente: `Alias url directorio`.

  Por ejemplo:

  ```
  Alias /docs /home/documentos
  ```

  provocará que una petición a http://www.uoc.edu/docs/manual se sirva desde /home/documentos/manual.

- **`UserDir`**: esta directiva nos permite indicar a Apache que un subdirectorio del directorio de trabajo de los usuarios del sistema sirva para almacenar su página personal.

  Por ejemplo:

  ```
  UserDir publico
  ```

  provocará que la página almacenada en el directorio del usuario `test`, en el subdirectorio público, sea accesible como:

  ```
  http://www.uoc.edu/~test/indice.html
  ```

Directivas de sección

La mayor parte de secciones de localización (`Directory`, `Location`, etc.) incluyen en su configuración una serie de directivas

ANOTACIONES

que nos permiten controlar el acceso al contenido ubicado dentro. Dichas directivas vienen proporcionadas por el módulo `mod_access`.

- **`Allow`**: nos permite especificar quién está autorizado a acceder al recurso. Podemos especificar direcciones IP, nombres de máquina, fragmentos del nombre o dirección e incluso por variables de la petición. Disponemos de la palabra clave `all` para indicar todos los clientes.

- **`Deny`**: nos permite especificar a quién no dejamos acceder al recurso. Disponemos de las mismas opciones que con `Allow`.

- **`Order`**: nos permite afinar el funcionamiento de las directivas `Allow` y `Deny`. Disponemos de dos opciones:

  – `Allow,Deny`. El acceso está denegado por defecto y sólo podrán entrar los clientes que cumplan las especificaciones de Allow y no cumplan las de Deny.

  – `Deny,Allow`. El acceso está permitido por defecto y sólo podrán entrar los clientes que no cumplan las especificaciones de Deny o cumplan las de Allow.

Servidores virtuales

Apache soporta servir diversos sitios web con un sólo servidor. Para ello proporciona facilidades de creación de dominios virtuales en función de diversas direcciones IP o diversos nombres por IP.

Apache fue uno de los primeros servidores en soportar servidores virtuales sin IP, en función de nombre. Esta capacidad simplifica mucho la administración de los servidores, además de suponer un ahorro importante de direcciones IP, normalmente escasas. Los servidores virtuales por nombre son totalmente transparentes para el cliente, con la única posible excepción de aquellos navegadores muy antiguos que no envíen la cabecera `Host:` con las peticiones.

ANOTACIONES

- **Servidores virtuales por dirección IP**

Para atender a diversos servidores virtuales, cada uno con una dirección IP, utilizaremos la sección de configuración `VirtualHost`. Con esta sección definiremos cada uno de los servidores con su propia configuración y dirección IP.

Un ejemplo sería el siguiente.

```
<VirtualHost 192.168.1.1>
   ServerAdmin webmaster@uoc.edu
   DocumentRoot /web/uoc
   ServerName www.uoc.edu
   ErrorLog /web/logs/uoc_error_log
   TransferLog /web/logs/uoc_access_log
</VirtualHost>
<VirtualHost 192.168.254.254>
   ServerAdmin webmaster@asociados.uoc.edu
   DocumentRoot /web/asociados
   ServerName asociados.uoc.edu
   ErrorLog /web/logs/asociados_error_log
   TransferLog /web/logs/asociados_access_log
</VirtualHost>
```

Como podemos ver, este ejemplo define dos servidores web, cada cual con una IP y un nombre diferente. Ambos tienen su propio `DocumentRoot`, etc.

Para utilizar servidores virtuales por IP, es necesario que el sistema servidor tenga configuradas en el sistema operativo las diversas direcciones IP que hay que servir.

- **Servidores virtuales por nombre**

Para atender a diversos servidores, utilizando la misma dirección IP para todos, utilizaremos la sección `VirtualHost`, que nos permitirá definir los parámetros de cada servidor. Si tenemos las mismas necesidades que en el ejemplo de servidores virtuales por dirección IP con una sola dirección utilizaremos la siguiente configuración:

ANOTACIONES

```
NameVirtualHost *:80
   <VirtualHost *:80>
      ServerAdmin webmaster@uoc.edu
      ServerName www.uoc.edu
      DocumentRoot /web/uoc
      ErrorLog /web/logs/uoc_error_log
      TransferLog /web/logs/uoc_access_log
   </VirtualHost>
   <VirtualHost *:80>
      ServerAdmin webmaster@uoc.edu
      ServerName asociados.uoc.edu
      DocumentRoot /web/asociados
      ErrorLog /web/logs/asociados_error_log
      TransferLog /web/logs/asociados_access_log
   </VirtualHost>.
```

Podemos utilizar una dirección IP en lugar de *, lo cual nos permitiría asignar un grupo de servidores virtuales por nombre a esa IP y otro grupo a otra, por ejemplo.

Un uso especial de las directivas de servidores por nombre se precisa cuando nuestro servidor tiene dos direcciones IP, pero hemos asignado a las dos el mismo nombre. Por ejemplo, cuando dispone una conexión de red en la intranet y otra en Internet con el mismo nombre, caso en que podemos servir el mismo contenido de la siguiente forma:

```
   NameVirtualHost 192.168.1.1
   NameVirtualHost 172.20.30.40

   <VirtualHost 192.168.1.1 172.20.30.40>
      DocumentRoot /www/servidor1
      ServerName servidor.uoc.edu
      ServerAlias servidor
</VirtualHost>
```

Con esta configuración podremos servir la misma web hacia la intranet y la Internet. Es conveniente notar el uso de un alias para el servidor que nos permite no usar dominios en la intranet.

Disponemos, además, de una especificación de servidor virtual por defecto `_default_` que nos permite atender las peticiones que no sirve ningún otro.

ANOTACIONES

```
   <VirtualHost _default_>
      DocumentRoot /www/defecto
</VirtualHost>
```

Podemos usar la etiqueta `_default_` con un número de puerto para especificar servidores por defecto diferentes para cada puerto.

Apache permite además configuraciones mucho más complejas de servidores virtuales, especialmente útiles en casos de servidores masivos, etc. Una magnífica guía de referencia se encuentra en la web del proyecto Apache, con consejos y recetas útiles para configurar.

2.3. Otros servidores web de software libre

Existen multitud de servidores HTTP de código libre, pero casi todos ellos han quedado eclipsados por la fama de Apache. Algunos de estos servidores presentan características que les dotan de gran interés.

2.3.1. AOLServer

El servidor web AOLServer es el servidor web de código libre desarrollado por AOL (América Online, el proveedor de Internet más importante del mundo). AOL utiliza AOLServer como servidor principal de web para uno de los entornos web de mayor tráfico y uso de Internet. AOLServer es un servidor web multi-hebra, basado en TCL, y con muchas facilidades de uso en entornos de gran escala y sitios web dinámicos. Cabe destacar que todos los dominios y servidores web de AOL, más de doscientos, que soportan miles de usuarios, millones de conexiones, etc., funcionan gracias a AOLServer.

AOLServer tiene una amplia base de usuarios, gracias sobre todo a su integración con OpenACS, un sistema de gestión de contenidos muy potente, de código libre, desarrollado inicialmente por una empresa llamada ArsDigita y posteriormente liberado bajo licencia GPL. El binomio AOLServer-OpenACS constituye la infraestructura de pro-

ANOTACIONES

yectos web tan complejos y potentes como dotLRN (un campus virtual universitario de código libre).

2.3.2. Roxen y Caudium

Roxen es un servidor web publicado bajo licencia GNU por un grupo de desarrolladores suecos que posteriormente fundarían la empresa Roxen Internet Services. El servidor Roxen (que primeramente se había llamado Spider y después, Spinner) ha destacado siempre por la gran cantidad de funcionalidades que ofrece a los usuarios. Este servidor, desarrollado en el lenguaje de programación Pike, ofrece a los usuarios cientos de módulos que permiten desarrollar fácilmente sitios web muy ricos, dinámicos, etc., sin otra herramienta que el servidor Roxen. Las características principales de Roxen son:

- Multiplataforma, es decir, puede ejecutarse en multitud de plataformas: Windows, Linux, Solaris, MAC OS/X, etc.
- Es de código libre.
- Interfaz de administración vía web muy rica y fácil de usar.
- Soporte gráfico integrado que permite, con sólo algunas etiquetas de RXML (la extensión de HTML de Roxen), generar imágenes, títulos, gráficas, etc.
- Acceso a BBDD integrado, posibilita el acceso a PostgreSQL, Oracle, MySQL, etc.
- Base de datos MySQL integrada.
- Programación en el servidor con RXML, Java, Perl, PHP y CGI.
- Soporte de criptografía fuerte.
- Arquitectura modular que permite cargar y descargar extensiones del servidor con éste en marcha.
- Independencia de plataforma en los módulos desarrollados por el usuario.

ANOTACIONES

A mediados del 2000, a raíz de la aparición de la versión 2.0 de Roxen, que rompía la compatibilidad de éste con las versiones anteriores, especialmente la 1.3, que era la más usada, un grupo de desarrolladores que incluía a algunos de los fundadores de Roxen, inició un nuevo proyecto que partía de la versión 1.3 de Roxen para desarrollar un servidor web que mantuviese la compatibilidad con ésta. Dicho servidor web se denominó Caudium. En estos momentos las dos plataformas, Roxen y Caudium, gozan de buena salud, de buenas relaciones (los desarrolladores intentan mantener la compatibilidad entre los API de los dos sistemas) y cuentan, además, con una base de usuarios fiel.

Roxen es uno de los pocos casos en los que un excelente producto (ha sido siempre uno de los servidores web más estables, rápidos y con mayor número de prestaciones y facilidades) no ha triunfado, ya que siempre ha sido eclipsado por Apache.

2.3.3. thttpd

thttpd es un servidor HTTP extremadamente pequeño, muy rápido, portable y seguro. Tiene las mismas prestaciones que otros servidores convencionales, como Apache, aunque en casos de carga extrema su rendimiento es mucho más alto.

Su utilidad como servidor web de propósito general es más bien escasa, aunque su uso principal suele ser como servidor rápido de contenido estático, muchas veces como soporte de servidores Apache para servir contenido binario estático, como puedan ser imágenes u otros, dejando para el servidor Apache las páginas dinámicas o más complejas. Usado como auxiliar de Apache para servir contenido estático ha conseguido reducir la carga del servidor principal a una centésima parte.

2.3.4. Jetty

Jetty es un servidor web escrito totalmente en Java que incluye, además, un contenedor de Servlets. Tiene un tamaño reducido y un rendimiento alto, lo que lo ha convertido en uno de los preferidos para desarrollar productos embebidos que requieran un servidor HTTP. Si

ANOTACIONES

bien no suelen encontrarse muchos servidores Jetty funcionando por sí mismos, sí que suelen encontrarse como servidores web empotrados en productos como:

- Integrados con servidores de aplicaciones, como JBoss y Jonas.
- Integrados en el proyecto JTXA como base para el transporte HTTP.
- Integrados en productos como Tivoli de IBM, MQ de Sonic y SESM de Cisco, como servidor HTTP.
- En la mayoría de CD de demostración en libros sobre Java, Servlets, XML, etc.
- Ejecutándose en múltiples sistemas embebidos y ordenadores de mano.

2.4. Prácticas: instalación del servidor web

2.4.1. Enunciado

1. Descargad de Internet el código del servidor Apache e instaladlo en un subdirectorio del directorio de nuestro usuario. Debemos instalar la última versión disponible, además de asegurarnos de la correcta instalación de los siguientes módulos:
   - mod_access
   - mod_cgi
2. Configurad el servidor instalado previamente para que responda a las peticiones HTTP en el puerto 1234.
3. Configurad el servidor web para que sirva los documentos que se encuentran en el subdirectorio `web` del directorio de trabajo de nuestro usuario.
4. Configurad el servidor web para que ejecute programas CGI del directorio `cgi` del directorio de trabajo de nuestro usuario.

ANOTACIONES

2.4.2. Resolución

1. Una vez obtenido el código fuente de Apache procedemos a descomprimirlo:

```
   [carlesm@bofh m2]$ tar xvzf httpd-2.0.48.tar.gz
httpd-2.0.48/
httpd-2.0.48/os/
httpd-2.0.48/os/os2/
httpd-2.0.48/os/os2/os.h
httpd-2.0.48/os/os2/core.mk
httpd-2.0.48/os/os2/config.m4
httpd-2.0.48/os/os2/Makefile.in
httpd-2.0.48/os/os2/core_header.def
....
httpd-2.0.48/include/ap_release.h
httpd-2.0.48/include/.indent.pro
httpd-2.0.48/include/util_cfgtree.h
httpd-2.0.48/acconfig.h
   [carlesm@bofh m2]$
```

Una vez tengamos el código fuente del servidor en nuestro directorio, podemos configurarlo para compilarlo. En este punto, le indicaremos a Apache dónde queremos instalarlo. En nuestro caso hemos escogido el subdirectorio `apache` de nuestro directorio de trabajo.

Además, nos aseguraremos de que se incluyen los módulos deseados.

```
[carlesm@bofh m2]$ cd httpd-2.0.48
[carlesm@bofh httpd-2.0.48]$ ./configure \
          --prefix=/home/carlesm/apache \.
          --enable-cgi
checking for chosen layout... Apache
checking for working mkdir -p... yes
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
....
creating srclib/pcre/Makefile
creating test/Makefile
```

ANOTACIONES

```
config.status: creating docs/conf/httpd-std.conf
config.status: creating docs/conf/ssl-std.conf
config.status: creating include/ap_config_layout.h
config.status: creating support/apxs
config.status: creating support/apachectl
config.status: creating support/dbmmanage
config.status: creating support/envvars-std
config.status: creating support/log_server_status
config.status: creating support/logresolve.pl
config.status: creating support/phf_abuse_log.cgi
config.status: creating support/split-logfile
config.status: creating build/rules.mk
config.status: creating include/ap_config_auto.h
config.status: executing default commands
[carlesm@bofh httpd-2.0.48]$
```

A continuación, ha llegado el momento de compilar.

```
   [carlesm@bofh httpd-2.0.48]$ make
Making all in srclib
make[1]: Entering directory '/srcs/httpd-2.0.48/srclib'
Making all in apr
make[2]: Entering directory '/srcs/httpd-2.0.48/srclib/apr'
Making all in strings
....
make[1]: Leaving directory '/srcs/httpd-2.0.48'
```

Si la compilación ha ido bien, podemos instalar Apache en el directorio escogido:

```
   [carlesm@bofh httpd-2.0.48]$ make install
Making install in srclib
make[1]: Entering directory '/srcs/httpd-2.0.48/srclib'
Making install in apr
make[2]: Entering directory '/srcs/httpd-2.0.48/srclib/apr'
Making all in strings
....
mkdir /home/carlesm/apache/man
mkdir /home/carlesm/apache/man/man1
mkdir /home/carlesm/apache/man/man8
mkdir /home/carlesm/apache/manual
Installing build system files
make[1]: Leaving directory '/srcs/httpd-2.0.48'
```

ANOTACIONES

```
   [carlesm@bofh httpd-2.0.48]$ cd /home/carlesm/apache/
   [carlesm@bofh apache]$ ls
bin      build      cgi-bin conf   error  htdocs
icons   include     lib     logs  man    manual
modules
   [carlesm@bofh apache]$
```

Seguidamente deberemos configurar Apache con los parámetros solicitados. Editamos para ello el fichero `/home/carlesm/apache/conf/httpd.conf` y modificamos los siguientes parámetros:

```
Listen 1234
ServerAdmin admin@uoc.edu
DocumentRoot "/home/carlesm/web"
<Directory "/home/carlesm/web">
   Options Indexes FollowSymLinks
   AllowOverride None
   Order allow,deny
   Allow from all
</Directory>
ScriptAlias /cgi-bin/ "/home/carlesm/cgi/"
<Directory "/home/carlesm/cgi">
   AllowOverride None
   Options None
   Order allow,deny
   Allow from all
</Directory>
```

ANOTACIONES

3. Diseño de páginas web

El lenguaje HTML (*hypertext markup language*) se utiliza para crear documentos que muestren una estructura de hipertexto. Un documento de hipertexto es aquel que contiene información cruzada con otros documentos, lo cual nos permite pasar de un documento al referenciado desde la misma aplicación con la que lo estamos visualizando. HTML permite, además, crear documentos de tipo multimedia, es decir, que contengan información más allá de la simplemente textual, como por ejemplo:

Ejemplo

- Imágenes
- Vídeo
- Sonido
- Subprogramas activos (*plug-ins*, *applets*)

El lenguaje HTML no es el único lenguaje existente para crear documentos hipertexto. Hay otros lenguajes anteriores o posteriores a HTML (SGML, XML, etc.), si bien HTML se ha convertido en el lenguaje estándar para la creación de contenido para Internet.

3.1. HTML básico

Los documentos HTML se conforman como documentos de texto plano (sin ningún tipo de formateo especial), en los que todo el formato del texto se especifica mediante marcas de texto (llamados etiquetas, *tags*), que delimitan los contenidos a los que afecta la etiqueta (disponemos de etiquetas de inicio y de final de marcado).

Las etiquetas o *tags* son marcas de texto que empiezan por el carácter <, seguido del nombre de la etiqueta, los atributos adicio-

ANOTACIONES

nales y acaban con el carácter >, de la forma, para las etiquetas de inicio:

```
<ETIQUETA>
```

Y que se forman con el carácter <, seguido del carácter /, seguido del nombre de la etiqueta y acaban con el carácter >, de la forma, para las etiquetas de final de marcado:

```
</ETIQUETA>
```

Las etiquetas no son sensibles a mayúsculas/minúsculas (son *case-insensitive*).

Ejemplos de etiquetas de HTML son:

```
<title>Nombre del documento</title>
<P>Un ejemplo de uso de las etiquetas para marcado de texto</P>
<B>Negrilla<I>Itálica</I>Negrilla</B>
```

Los atributos de las etiquetas, que especifican parámetros adicionales a la etiqueta, se incluyen en la etiqueta de inicio de la siguiente forma:

```
<ETIQUETA ATRIBUTO ATRIBUTO ...>
```

La forma de dichos atributos será bien el nombre del atributo o bien el nombre del atributo, seguido de =, seguido del valor que se le quiere asignar (generalmente entrecomillado).

Por ejemplo:

```
<A HREF="http://www.w3c.org">Enlace</A>
<IMG SRC="imagen.jpg" BORDER=0 ALT="NOMBRE">
```

Cabe destacar que HTML permite omitir en algunos casos la etiqueta de final si la etiqueta no debe envolver ningún texto al que afecte,

ANOTACIONES

como es el caso de IMG. Otro punto destacable es que si el cliente WWW que utilicemos (el navegador concreto que estemos usando) no entiende alguna etiqueta, la ignorará y hará lo mismo con todo el texto afectado por ésta.

3.1.1. Estructura de los documentos HTML

Todos los documentos HTML siguen aproximadamente la misma estructura. Todo el documento debe ir contenido en una etiqueta `HTML`, dividiéndose en dos partes: la cabecera, contenida en una etiqueta `HEAD` y el cuerpo del documento (donde está la información del documento), que está envuelto por una etiqueta `BODY`. La cabecera contiene algunas definiciones sobre el documento: su título, marcas extra de formato, palabras clave, etc.

Un primer ejemplo sería:

```
<HTML>
  <HEAD>
    <TITLE>Título del documento</TITLE>
  </HEAD>
  <BODY>
    Texto del documento
  </BODY>
</HTML>
```

Si abrimos un documento con este contenido en un navegador apreciaremos que lo que contienen las etiquetas `TITLE` no se visualiza en el documento, sino que el navegador las visualiza en la barra de título de la ventana.

Comentarios

En HTML podemos introducir comentarios en la página con las marcas <!--y --!>. El navegador ignora el contenido que se encuentre entre estas dos marcas y no lo muestra al usuario.

ANOTACIONES

ANOTACIONES

3.1.2. Bloques de texto

Disponemos de diversos tipos de bloques de texto en HTML:

- Párrafos.
- Saltos de línea.
- Bloques citados.
- Divisiones.
- Texto preformateado.
- Centrados.

Párrafos

Disponemos de la etiqueta <P> para separar párrafos. Habida cuenta de que HTML ignora los saltos de línea introducidos en el fichero original y de que para HTML todo el texto es continuo, requeriremos algún mecanismo para indicar principio y final de párrafo. Dicho mecanismo viene dado por las marcas <P> y </P>.

La etiqueta P admite además un atributo, ALIGN, que indica la alineación del texto en el párrafo, pudiendo tomar los valores:

- LEFT, alineación a la izquierda, es el activo por defecto.
- RIGHT, alineación a la derecha.
- CENTER, centrado del texto.

La marca de final de párrafo, </P>, está definida en el estándar de HTML como opcional, pudiéndose omitir. En ese caso, el navegador inferirá que una nueva aparición de <P> indica el final del párrafo anterior.

Saltos de línea

La etiqueta
 nos indica un salto de línea. Podemos usarla como marca inicial sin final. La marca BR no modifica los parámetros especificados para el párrafo en el que nos encontremos en ese momento.

Reglas horizontales

HTML proporciona una etiqueta que nos permite incluir en nuestra página una regla horizontal (una raya de extremo a extremo de la

página) de anchura variable. Dicha etiqueta, HR, se utiliza para separar bloques de texto. Es un elemento que sólo tiene etiqueta inicial, pero que dispone de diversos atributos para cambiar su forma:

- NOSHADE: elimina el efecto de sombreado de la regla.
- WIDTH: define la longitud de la línea respecto a la página.
- SIZE: define el grosor de la línea.

Párrafos citados

Disponemos de un elemento en HTML, BLOCKQUOTE, que permite representar párrafos citados literalmente de otro texto. Generalmente se representan identados a derecha e izquierda y con un salto de párrafo antes y después del párrafo citado. Deberíamos evitar el uso de BLOCKQUOTE para identar texto, usándolo sólo para citas literales, pues el navegador puede representarlo de otra forma, por ejemplo, no identado.

Divisiones del texto en bloques

El elemento <DIV> permite dividir el texto en bloques, insertando entre los bloques un salto de línea simple, al igual que BR. No obstante, a diferencia de éste, admite los mismos atributos que P, es decir, podemos definir la alineación del texto para cada bloque DIV.

Las alineaciones soportadas por DIV, mediante el parámetro ALIGN, son las siguientes:

- LEFT, alineación a la izquierda, es el activo por defecto.
- RIGHT, alineación a la derecha.
- CENTER, centrado del texto.

Texto preformateado

El texto insertado entre las etiquetas <PRE> y </PRE> será visualizado por el navegador respetando el formato de saltos de línea y espacios con el que se ha introducido. El navegador, por norma

ANOTACIONES

general, mostrará dicho texto con un tipo de letra de espaciado fijo similar a la de las máquinas de escribir.

Podemos ver algunas de estas etiquetas en el siguiente ejemplo:

```
<HTML>
   <HEAD>
      <TITLE>Título del documento</TITLE>
   </HEAD>
   <BODY>
<P ALIGN=LEFT>
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y galgo corredor. Una
olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lantejas los viernes, algún
palomino de añadidura los domingos, consumían las tres
partes de su hacienda.
</P>
<DIV ALIGN=RIGHT>
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y galgo corredor. Una
olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lantejas los viernes, algún
palomino de añadidura los domingos, consumían las tres
partes de su hacienda.
</DIV>
<DIV ALIGN=CENTER>
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y galgo corredor. Una
olla de algo más vaca que carnero, salpicón las más noches,
duelos y quebrantos los sábados, lantejas los viernes, algún
palomino de añadidura los domingos, consumían las tres
partes de su hacienda.
</DIV>
<PRE>
En un lugar de la Mancha,
              de cuyo nombre no quiero acordarme,
   no ha mucho tiempo que vivía un hidalgo
           de los de lanza en astillero, adarga
           antigua, rocín flaco y galgo corredor.
Una olla de algo más vaca que carnero,
</PRE>
```

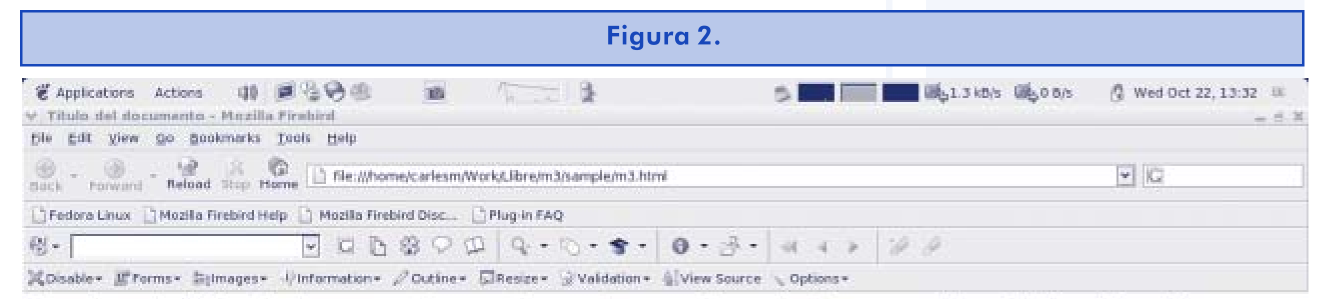
ANOTACIONES

```
<ADDRESS>
   Miguel de Cervantes<BR>
   Calle Shakespeare, 23<BR>
   09876, Madrid<BR>
</ADDRESS>

   <CENTER>
      <P>
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
no ha mucho tiempo que vivía un hidalgo de los de lanza
en astillero,
       </P>
    </CENTER>
</BODY>
</HTML>
```

El resultado de la visualización de este código HTML es el siguiente:

Figura 2.



Una de las utilidades que nos proporcionan algunos navegadores de código libre, como por ejemplo Mozilla o Firebird, es la de mostrarnos los elementos de bloque que componen una página web. En este caso, el ejemplo se mostraría de la siguiente manera:

Figura 3.



3.1.3. Marcadores lógicos

HTML presenta además un grupo de etiquetas que permiten dar formato al texto no en función de cómo queremos que se represente, sino en función de la semántica de ese bloque de texto, permitiendo así al navegador representar el texto en pantalla de la forma más conveniente.

Estas etiquetas son:

- <CITE>: cita literal de texto o documento.
- <ADDRESS>: dirección.
- <SAMP>: ejemplo de código o resultado.
- <CODE>: código de programa.
- <KBD>: datos que deben teclearse.
- <VAR>: definición de variable.
- <DFN>: definición de texto o palabra (soporte no muy extendido entre los navegadores).

Dichas etiquetas serán formateadas de forma diferente según el navegador y la configuración que tengamos. Como ejemplo podemos ver el aspecto que presentan en el navegador Mozilla:

Figura 4.

The C Programming Language, Ritchie, Dennis; Kernighan, Ritchie. AT&T Bell Labs.

Nuestra dirección es:

10, Downing Street. London.

Los ficheros que acaban con la extensión `jpg` corresponden a imágenes.

`printf("Hello World\n");`

Una vez entrado al sistema teclear `startx` para arrancar...

Definiremos la variable *neigh* para guardar...

Un *Distributed-CSP* es un problema, la solución del ...

© 2003, Carles Mateu

El código que ha producido este resultado es el siguiente:

```
<HTML>
   <HEAD>
      <TITLE>Título del documento</TITLE>
   </HEAD>
   <BODY>
```

```
<P><CITE>The C Programming Language</CITE>,
Ritchie, Dennis; Kernighan, Ritchie. AT&T Bell Labs.

<P> Nuestra dirección es:
<ADDRESS>
10, Downing Street.
Londres.
</ADDRESS>

<P>
Los ficheros que acaban con la extensión
<SAMP>jpg</SAMP> corresponden a imágenes.

<P>
<CODE>printf("Hello World\n");</CODE>

<P>Una vez entrado teclear <KBD>startx</KBD> para arrancar...

<P>Definiremos la variable <VAR>neigh</VAR> para guardar...

<P>Un <DFN>Distributed-CSP</DFN> es un problema, del ...


<P><CITE>&#169; 2003, Carles Mateu</CITE>

   </BODY>
</HTML>
```

3.1.4. Tipos de letra

HTML proporciona etiquetas que nos permiten modificar el tipo de letra, los colores, etc., de nuestros textos. Además, HTML proporciona unos marcadores especiales, llamados entidades de carácter, que permiten introducir caracteres especiales, como el símbolo de *copyright*, acentos, y otros, para aquellos casos en los que nuestro teclado o nuestro editor de texto no lo soportan, para cuando el juego de caracteres no lo soporta, etc.

Cabeceras

Disponemos de un elemento `<Hx>` que podemos utilizar para definir qué partes de nuestro texto deben ser consideradas como encabezados (de sección, capítulo, etc.). La etiqueta asigna un mayor tamaño de texto a los caracteres (en función de `x`, como veremos). Además,

ANOTACIONES

utiliza un tipo de letra en negrilla para la cabecera e incluye un salto de párrafo después de esta cabecera.

El tamaño de la cabecera (o el nivel de ésta o el índice de importancia) puede variar entre 1 y 6, es decir, existen 6 posibles etiquetas: `H1, H2, H3, H4, H5` y `H6`.

Tipo de letra

HTML proporciona una etiqueta para gestionar la tipografía. Esta etiqueta, `FONT`, ha sido declarada obsoleta en HTML 4.01, por lo que debemos evitar su uso y tratar de usar en su lugar hojas de estilo (CSS). `FONT` nos permite especificar:

- Medidas, con el atributo `SIZE`.
- Colores, con el atributo `COLOR`.
- Tipografías, con el atributo `FACE`.

Debemos utilizar esta etiqueta con mucha precaución. Al especificar tipografías debemos tener en cuenta que quizás el cliente no tenga dicha tipografía instalada, con lo que la visión que obtendrá de la página no corresponderá a la deseada.

Los atributos soportados por `FONT` y que nos permiten definir las características del tipo de letra son:

- `SIZE`: tamaño de los caracteres, con valores de 1 a 7, o valores relativos (-7 a +7).
- `COLOR`: color de los caracteres.
- `FACE`: tipografía que vamos a usar. Podemos especificar más de una, separadas por comas.

El atributo `SIZE` define el tamaño de letra en función del tamaño definido por defecto para el documento, que definiremos con `BASEFONT`. `BASEFONT` tiene un solo parámetro, `SIZE`, que nos permite definir el tamaño base del documento.

ANOTACIONES

Estilos de letra

HTML nos proporciona un conjunto de etiquetas que permiten definir diferentes estilos de letras para el texto contenido entre las marcas. Las etiquetas disponibles son:

`B` (negrilla).
`I` (itálica).
`U` (subrayado).
`STRIKE` (tachado).
`SUP` (superíndice).
`SUB` (subíndice).
`BLINK` (parpadeo).
`TT` (teletipo).
`BIG` (grande).
`SMALL` (pequeño).

Además de estos tipos físicos de letra, disponemos también de algunos tipos lógicos de letra, que los navegadores pueden preferir representar de otra forma:

`EM` (enfatizado).
`STRONG` (destacado).
`CODE` (código de programa).
`CITE` (cita).
`KBD` (entrada por teclado).
`SAMP` (ejemplos).
`VAR` (variables de programa).

Algunos de estos estilos lógicos suponen asimismo un estilo de párrafo. Éstos ya se han estudiado anteriormente.

HTML permite mezclar diferentes estilos, como negrilla y cursiva (itálica), etc. En ese caso se anidarían las etiquetas de HTML correspondientes:

```
<B><I>Negrilla y cursiva</I></B>
```

ANOTACIONES

Podemos ver el aspecto de los tipos de letras y colores en la siguiente página. El código HTML que ha producido este resultado es el siguiente:

Figura 5.

Cabecera H1

Cabecera H2

Cabecera H3

Cabecera H4

Cabecera H5

Cabecera H6

Tamaño de letra

1 2 3 4 5 6 7 6 5 4 3 2 1

Colores COLORES.DE.LETRAS

Negrita
Cursiva
Subrayado
~~Tachado~~
A Superíndice
B Subíndice

`Máquina de escribir(Teletipo)`
Texto grande
Texto pequeño

ANOTACIONES

```
<HTML>
   <HEAD>
      <TITLE>Título del documento</TITLE>
   </HEAD>
<BODY>

<h1>Cabecera H1</h1>
<h2>Cabecera H2</h2>
<h3>Cabecera H3</h3>
<h4>Cabecera H4</h4>
<h5>Cabecera H5</h5>
<h6>Cabecera H6</h6>

<b>Tamaño de letra</b> <BR>
<font SIZE=1>1</font> <font SIZE=2>2</font>
<font SIZE=3>3</font> <font SIZE=4>4</font>
<font SIZE=5>5</font> <font SIZE=6>6</font>
<font SIZE=7>7</font> <font SIZE=6>6</font>
<font SIZE=5>5</font> <font SIZE=4>4</font>
<font SIZE=3>3</font> <font SIZE=2>2</font>
<font SIZE=1>1</font>
<P>
<B>Colores</b>
<font COLOR=#800000>C</font><font COLOR=#000080>O</font>
<font COLOR=#000080>L</font><font COLOR=#008000>O</font>
<font COLOR=#00FFFF>R</font><font COLOR=#FF0000>E</font>
<font COLOR=#C0C0C0>S</font> . <font COLOR=#800080>D</font>
<font COLOR=#008080>E</font> . <font COLOR=#FF0000>L</font>
<font COLOR=#808080>E</font><font COLOR=#FF00FF>T</font>
<font COLOR=#00FF00>R</font><font COLOR=#808000>A</font>
<font COLOR=#FFFF00>S </font>

<P> <b>Negrilla</b> <br> <i>Cursiva</i> <br> <u>Subrayado</u><br>
<strike>Tachado</strike> <br> A<sup>Super&iacute;ndice</sup> <br>
B<sub>Sub&iacute;ndice</sub><br> <blink>Parpadeo</blink> <br>
<tt>M&aacute;quina de escribir(Teletipo)</tt> <BR> <big>Texto
grande</big> <br> <small>Texto pequeño</small>


</BODY>
</HTML>
```

ANOTACIONES

Entidades de carácter

HTML proporciona una serie de códigos especiales, llamados entidades de carácter, que nos permiten introducir caracteres que no podamos meter desde el teclado, como acentos, circunflejos, símbolos especiales, etc. Además, podemos introducir cualquier carácter de la tabla de caracteres de ISO-Latín1 con una entidad de carácter especial.

Tabla 2. Tabla de códigos

Código	Resultado
á, Á, é, É,...	á, á, é, é,...
¿	¿
¡	¡
º	º
ª	ª
™ o ™	Símbolo de TradeMark
©	Símbolo de Copyright
®	Símbolo de Registrado
	(espacio en blanco que no puede ser usado para saltar de línea)
<	<
>	>
&	&
"	"

3.1.5. Enlaces

Una de las características más destacadas y de mayor influencia en el éxito de la web ha sido su carácter hipertexto, es decir, la posibilidad de que podamos enlazar documentos que pueden residir en diferentes servidores de forma intuitiva y transparente. Cabe destacar que los enlaces no sólo llevan a páginas web, sino también a imágenes, audio, vídeo, etc.

Disponemos para realizar los enlaces de una etiqueta A, que con su conjunto de atributos, `NAME`, `HREF`, `TARGET`, nos permite un total control a la hora de crear enlaces en los documentos.

ANOTACIONES

Disponemos de cuatro tipos de enlaces principales:

- Enlaces dentro de la misma página.
- Enlaces a otras páginas de nuestro sistema.
- Enlaces a páginas de otros sistemas.
- Enlaces documentos consultados a través de otros protocolos (correo electrónico, etc.).

Los enlaces

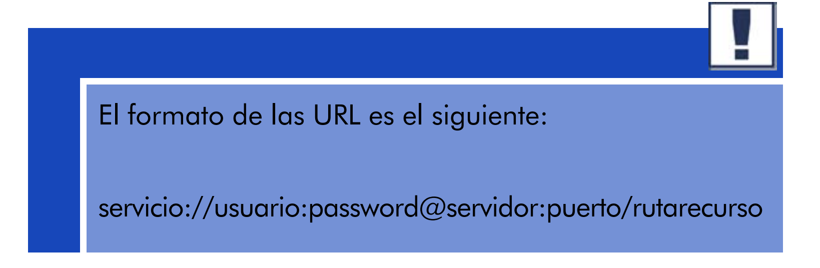
Para crear un enlace utilizaremos la etiqueta `A` con el atributo `HREF`. Como valor de dicho atributo pasaremos el destino del enlace:

```
<A HREF="destino">Texto o imagen</A>
```

El contenido de la etiqueta se considerará como especial. Será visualizado de forma diferente por el navegador (generalmente subrayado). Al pulsar sobre él saltaremos al destino indicado por el valor del atributo `HREF`, que debe ser una URL.

Los destinos

Un destino es una dirección URL que nos indica un servicio que deseamos obtener o un recurso al que deseamos acceder.



Existen diversos servicios que podemos indicar en la URL y que serán aceptados por la mayoría de navegadores:

- http: indica el servicio de transferencia de páginas web y es el usado habitualmente.

- https: indica el servicio HTTP seguro y cifrado.

- ftp: indica que debemos usar el protocolo de transferencia de archivos, `FTP`. Si no indicamos usuario y contraseña, se intentará la transferencia anónima. En caso de que ésta no fuese aceptada, se nos pediría usuario y contraseña.

- mailto: indica que se debe enviar un correo electrónico a la dirección especificada.

- news: acceso al servicio de News USENET.

Ejemplo

Algunos ejemplos de URL son:

http://www.uoc.edu
https://www.personales.co/usuarios/carles/indice.html
ftp://usuario:secreto@ftp.cesca.es/pub/linux
mailto:destino@correo.electroni.co
news://noticias.uoc.edu/es.comp.os.linux

Destinos dentro de la página

Una de las posibilidades que nos ofrece HTML consiste en saltar a destinos dentro de una página. Para ello debemos definir en la página los destinos con nombre, *anchors* o anclas. Para realizar dicha definición, disponemos del atributo `NAME` de la etiqueta A.

Por ejemplo:

```
<A NAME="ancla">
```

Una vez definidas las anclas dentro de nuestros documentos, podemos navegar e ir directamente a ellas. Para navegar a estas anclas usaremos una extensión de la URL.

Por ejemplo:

```
<A HREF="http://www.uoc.edu/manual.html\#ancla))Enlace</A>
```

ANOTACIONES

Si el enlace lo hiciésemos dentro de la misma página, podríamos abreviar la dirección con:

```
<A HREF="\#ancla))Enlace</A>
```

3.1.6. Listas

HTML nos permite definir listas y enumeraciones de tres tipos principales:

- Listas no ordenadas.
- Listas ordenadas (numeradas).
- Listas de definiciones.

Listas no ordenadas

Para introducir listas no ordenadas disponemos de la etiqueta `<UL>`, que nos indica el inicio de la lista, de `</UL>`, que nos indicará el final de la lista y de `<LI>`, que nos indica cada uno de los elementos de la lista.

Disponemos además de un atributo `TYPE` que nos indica el marcador que hay que usar para señalar los diferentes elementos: `DISC`, `CIRCLE`, `SQUARE`.

Debemos introducir todos los elementos entre `<LI>` y `</LI>`.

Listas ordenadas (numeradas)

El funcionamiento de las listas ordenadas es muy similar al de las listas no ordenadas. Para éstas disponemos de la etiqueta `<OL>`, que nos indica el inicio de la lista, de `</OL>` que nos indicará el final de la lista y de `<LI>` que nos indica cada uno de los elementos de la misma.

ANOTACIONES

Disponemos, además, de un atributo `TYPE` que nos indica el marcador que conviene usar para enumerar los diferentes elementos:

`TYPE=1` Números (la elegida por defecto).
`TYPE=A` Letras mayúsculas.
`TYPE=a` Letras minúsculas.
`TYPE=I` Numeración romana en mayúsculas.
`TYPE=i` Numeración romana en minúsculas.

Disponemos también de un atributo `START` que nos indica en qué punto debe comenzar la numeración de la línea. El atributo `TYPE` también puede usarse en los elementos individuales.

Debemos introducir todos los elementos entre `<LI>` y `</LI>`.

Listas de definiciones

Una lista de definiciones es una lista no numerada que nos permite dar una descripción o definición de cada elemento. Las listas descriptivas están formadas con las etiquetas: `<DL>` y `</DL>` para definir la lista, `<DT>` para indicar el término que hay que definir y `DD` para indicar la definición.

Disponemos para `DL` de un atributo, `COMPACT`, que indica al navegador que muestre la lista de la forma más compacta posible, incluyendo el término y su definición en la misma línea.

Podemos ver diferentes ejemplos de listas de HTML:

Figura 6.

- Primer elemento
- Segundo elemento
- Tercer elemento

1. Primer elemento
2. Segundo elemento
C. Tercer elemento

ASCII
    Juego de caracteres de 7 bits. Solo permite 127 caracteres.
EPS
    Formato Postscript encapsulado.
PNG
    Portable Network Graphics, formato gráfico de alta eficiencia.

ANOTACIONES

El código HTML que ha producido este resultado es el siguiente:

```
<HTML>
 <HEAD>
 <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>


<UL>
 <LI>Primer elemento
 <LI>Segundo elemento
 <LI>Tercer elemento
</UL>
<P>


<OL>
 <LI>Primer elemento
 <LI>Segundo elemento
 <LI TYPE=A>Tercer elemento
</OL>
<P>


<dl compact>
 <dt>ASCII <dd>
Juego de caracteres de 7 bits.
Sólo permite 127 caracteres.
 <dt>EPS <dd>
Formato Postscript encapsulado.
 <dt>PNG<dd>
Portable Network Graphics,
formato gráfico de alta eficiencia.
</dl>

</BODY>
</HTML>
```

ANOTACIONES

3.1.7. Imágenes

Para poder incluir imágenes y gráficos en nuestras páginas disponemos de una única etiqueta: `<IMG>`.

`<IMG>` dispone de diversos atributos que nos permiten especificar qué fichero de imagen queremos usar, las medidas de ésta, etc. El atributo que nos permite especificar qué imagen debemos mostrar es `SRC`. Con dicha etiqueta podemos especificar una URL de fichero de imagen que será el que el navegador solicitará al servidor para mostrárnoslo.

Las imágenes que referenciemos con el atributo `SRC` pueden estar en cualquier directorio del servidor, en otros servidores, etc. El valor que pasemos a `SRC` debe ser una URL.

Disponemos, además, de un atributo `ALT` que nos permite asignar un texto alternativo a la imagen para el caso de que el navegador no pudiese mostrar la imagen que enseñase ese texto al usuario.

Disponemos asimismo de un par de atributos que nos permiten especificar las medidas de la imagen, anchura y altura, `WIDTH` y `HEIGHT`. En caso de no especificarlos, el navegador mostrará la imagen con el tamaño que tenga el archivo de ésta. En caso de especificarlos, el navegador redimensionará la imagen adecuadamente. Usar los parámetros de medida de imagen permite al navegador dejar el espacio que ocupará la imagen y mostrar el resto de la página mientras se descarga éstas.

Un uso habitual de las imágenes es como botones para enlaces, en cuyo caso el navegador generalmente le añadirá un borde para diferenciarlo del resto. Podemos evitar este efecto añadiendo un atributo adicional, `BORDER`, que nos permite especificar el grosor de este borde o eliminarlo poniéndolo a cero.

Figura 7.



El código HTML que ha producido la pantalla es el siguiente:

```
<HTML>
 <HEAD>
    <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>

<IMG SRC="logo.gif"> <P>

<IMG SRC="nologo.gif" ALT="IMAGEN NO EXISTENTE"><P>

</BODY>
</HTML>
```

3.1.8. Tablas

Disponemos en HTML de un grupo de etiquetas que nos permite introducir texto en forma de tablas. Las etiquetas para ello son:

`TABLE`: marca el inicio y final de la tabla.

`TR`: marca el inicio y final de una fila.

`TH`: marca el inicio y final de una celda de cabecera.

`TD`: marca el inicio y final de una celda.

`CAPTION`: permite insertar títulos en las tablas.

Una tabla simple queda conformada por el siguiente código:

```
<TABLE>
 <TR><TH>Cabecera 1</TH>...<TH>Cabecera n</TH></TR>
 <TR><TD>Celda 1.1</TD>...<TD>Celda n</TD></TR>

 ...

 <TR><TD>Celda 1.1</TD>...<TD>Celda n</TD></TR>
 <CAPTION>Título</CAPTION>
</TABLE>
```

ANOTACIONES

Como podemos ver, la tabla queda envuelta dentro de una etiqueta TABLE. Para cada fila de la tabla debemos envolver esta fila entre las etiquetas <TR> y </TR>. En cada fila tenemos dos opciones para mostrar las celdas: podemos envolverlas entre etiquetas <TH> o entre etiquetas <TD>. La diferencia entre ellas es que la primera selecciona un tipo de letra negrilla y centra la columna.

Etiqueta <TABLE>

La etiqueta TABLE tiene algunos atributos que nos permiten especificar más concretamente el formato que queremos darle a la tabla.

BORDER: nos indica el tamaño de los bordes de la celda.

CELLSPACING: nos indica el tamaño en puntos del espacio entre celdas.

CELLPADDING: nos indica el tamaño en puntos entre el contenido de una celda y los bordes.

WIDTH: especifica el ancho de la tabla. Puede estar tanto en puntos como con relación al porcentaje de la anchura total disponible. Por ejemplo, 100% indica toda la ventana del navegador.

ALIGN: alinea la tabla respecto a la página, a la izquierda (LEFT), a la derecha (RIGHT) o al centro (CENTER).

BGCOLOR: especifica el color de fondo de la tabla.

Etiqueta <TR>

La etiqueta TR nos permite introducir las diferentes filas que componen una tabla. TR presenta los siguientes atributos:

ALIGN: alinea el contenido de las celdas de la fila horizontalmente a izquierda (LEFT), derecha (RIGHT) o centro (CENTER).

VALIGN: alinea el contenido de las celdas de la fila verticalmente arriba (TOP), abajo (BOTTOM) o centro (MIDDLE).

BGCOLOR: especifica el color de fondo de la fila.

ANOTACIONES

ANOTACIONES

Etiquetas <TD> y <TH>

Las etiquetas TD y TH nos permite introducir las celdas que formarán la fila en la que se encuentran. La diferencia principal entre ellas estriba en que TH centra horizontalmente el contenido de la celda y lo muestra en negrillas. Estas dos etiquetas pueden presentar los siguientes atributos:

ALIGN: alinea el contenido de las celdas de la fila horizontalmente a izquierda (LEFT), derecha (RIGHT) o centro (CENTER).

VALIGN: alinea el contenido de las celdas de la fila verticalmente arriba (TOP), abajo (BOTTOM) o centro (MIDDLE).

BGCOLOR: especifica el color de fondo de la celda.

WIDTH: especifica el ancho de la celda, en puntos o porcentualmente. En este último caso hay que tener en cuenta que se refiere al ancho de la tabla, no de la ventana.

NOWRAP: impide que en el interior de las celdas se divida la línea por espacios.

COLSPAN: especifica cuántas celdas, incluyendo esta, se unirán hacia la derecha para formar una sola. Si COLSPAN es cero, todas las celdas se unirán a la derecha.

ROWSPAN: especifica el número de celdas de la columna situadas debajo de la actual que se unen a ésta.

Etiqueta <CAPTION>

Nos permite añadir una leyenda o título centrado sobre o debajo de una tabla. Disponemos de un solo atributo:

ALIGN: nos indica dónde se situará el CAPTION respecto a la tabla, siendo los posibles valores: TOP, que la sitúa sobre la tabla y BOTTOM, que la sitúa debajo de ella.

Podemos ver en la imagen dos tablas de HTML:

Figura 8.

1,1 y 1,2		1,3
2,1 y 3,1	2,2	2,3
	3,2	3,3

Tabla Simple

UOC	Abril	Mayo	Junio	Julio
Vehiculos	22	23	3	29
Visitantes	1234	1537	7	1930
Ingresos	11000	13000	-500	60930

El código HTML que ha producido este resultado es el siguiente:

```
<HTML>
 <HEAD>
  <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>

<TABLE BORDER=1>
  <TR>
    <TD COLSPAN=2>1,1 y 1,2</TD>
    <TD>1,3</TD>
  </TR>
  <TR>
    <TD ROWSPAN=2>2,1 y 3,1</TD>
    <TD>2,2</TD>
    <TD>2,3</TD>
  </TR>
  <TR>
    <TD>3,2</TD>
    <TD>3,3</TD>
 </TR>
```

ANOTACIONES

```
<CAPTION ALIGN=bottom>Tabla  Simple</CAPTION>
</TABLE>

<HR>

<TABLE  BORDER=0  CELLSPACING=0  BGCOLOR=#0000FF>
<TR><TD>
<TABLE  BORDER=0  CELLSPACING=1  CELLPADDING=2
    WIDTH=400  BGCOLOR=#FFFFFF>
  <TR>
    <TH><IMG  SRC="logo.gif"></TH>
    <TH>Abril</TH>
    <TH>Mayo</TH>
    <TH>Junio</TH>
   <TH>Julio</TH>
  </TR>
  <TR>
    <TD  BGCOLOR=#A0A0A0>Vehículos</TD>
    <TD>22</TD>
    <TD>23</TD>.
    <TD  BGCOLOR=#FFa0a0>3</TD>
    <TD>29</TD>
  </TR>
  <TR>
    <TD  BGCOLOR=#A0A0A0>Visitantes</TD>
    <TD>1234</TD>
    <TD>1537</TD>
    <TD  BGCOLOR=#FFa0a0>7</TD>
    <TD>1930</TD>
  </TR>
  <TR>
    <TD  BGCOLOR=#A0A0A0>Ingresos</TD>
    <TD>11000</TD>
    <TD>13000</TD>
    <TD  BGCOLOR=#FF4040>-500</TD>
    <TD  BGCOLOR=#a0a0FF>60930</TD>
  </TR>
</TABLE>
</TD></TR>
</TABLE>
</BODY>
</HTML>
```

ANOTACIONES

3.1.9. Formularios

Los formularios son elementos de HTML que permiten recoger información del usuario. Disponemos de una gran variedad de elementos de formulario que nos permiten interactuar de forma rica y eficiente con los usuarios. De todas maneras, cabe destacar que los formularios no procesan la información introducida por los usuarios, sino que debemos procesarla nosotros por otros medios (CGI, JSP, Servlets, etc.).

Un ejemplo de cómo crear un formulario es el siguiente:

```
<FORM ACTION="url proceso" METHOD="POST">
...
Elementos
..
</FORM>
```

La etiqueta `FORM` nos proporciona algunos atributos:

- `ACTION`: este atributo nos permite especificar a qué URL se enviarán los datos que el usuario haya introducido en el formulario. Podemos especificar como URL una dirección de correo electrónico, de la forma:

  `mailto:direccion@correo.electron`

  o podemos especificar una URL de HTTP (el método más usado para enviar los datos a los programas de tipo CGI):

  `http://www.uoc.edu/proceso.cgi`

- `METHOD`: el método especifica de qué forma se envían los datos. Disponemos de dos opciones: `GET` y `POST`. Veremos con más detalle el funcionamiento de estas dos opciones al hablar de programación CGI.

- `ENCTYPE`: especifica el tipo de codificación usada. Generalmente sólo se usa para especificar en caso de enviar el resultado del formulario por correo para cambiar la codificación a `text/plain`.

- `NAME`: sirve para asignar un nombre al formulario, necesario pasa usarlo posteriormente desde Javascript.

ANOTACIONES

Elementos del formulario

HTML proporciona una gran variedad de elementos de entrada para los formularios. Estos elementos permiten desde introducir texto hasta enviar ficheros.

- Elemento <**INPUT**>

El elemento INPUT es quizás el más conocido y usado de los elementos de los formularios. Actúa como un campo de entrada, disponiendo de diversos tipos de elementos INPUT en función del valor del atributo TYPE:

– TYPE=RADIO: permite escoger entre múltiples opciones, pero sólo una de las del mismo nombre.

– TYPE=RESET: pone en blanco todo el formulario.

– TYPE=TEXT: permite la entrada de una línea de texto.

– TYPE=PASSWORD: permite la entrada de una línea de texto mostrando en lugar de éste caracteres como "*", usado generalmente para entrada de contraseñas.

– TYPE=CHECKBOX: permite escoger una o varias opciones múltiples.

– TYPE=SUBMIT: acepta los datos entrados en el formulario y ejecuta la acción especificada.

– TYPE=HIDDEN: campo de texto no visible para el usuario. Usado para conservar valores.

El elemento INPUT dispone, además, de otros atributos opcionales:

– NAME: da nombre al campo. Es importante ponerlo para después poder procesarlo desde nuestros programas.

– VALUE: otorga un valor inicial al campo.

– SIZE: tamaño en el caso de los campos TEXT y PASSWORD.

– MAXLENGTH: longitud máxima aceptada de la entrada del usuario (campos TEXT y PASSWORD).

ANOTACIONES

– CHECKED: en caso de RADIO o CHECKBOX si ésta está marcada o no por defecto.

• Elemento **SELECT**

El elemento SELECT nos permite seleccionar una o varias de las opciones presentadas.

Un ejemplo de elemento SELECT sería:

```
<SELECT name="destino">
     <option> África
     <option> Antártida
     <option> América
     <option> Asia
     <option> Europa
     <option> Oceanía
</SELECT>
```

Los atributos que ofrece el elemento SELECT son:

– SIZE: el tamaño en pantalla del elemento SELECT. Si es uno, sólo nos mostrará una opción y el elemento SELECT actuará como una lista desplegable. Si es mayor que uno, veremos una lista de selección.

– MULTIPLE: si lo indicamos, podremos elegir más de una opción.

El elemento OPTION dispone de dos atributos:

– VALUE: el valor que se asignará a la variable al seleccionar esta opción.

– SELECTED: qué opción quedará seleccionada por defecto.

• Elemento **TEXTAREA**

El elemento TEXTAREA nos permite recoger del usuario elementos de texto de múltiples líneas. El formato es el siguiente:

```
<TEXTAREA name="comentarios" cols=30 rows=6>
Introduzca comentarios sobre la página
</TEXTAREA>
```

ANOTACIONES

Cabe destacar que el contenido encerrado entre `<TEXTAREA>` y `</TEXTAREA>` será considerado el valor inicial del campo. Los atributos que nos ofrece `TEXTAREA` son:

– `ROWS`: filas que ocupará la caja de texto.
– `COLS`: columnas que ocupará la caja de texto.

Podemos ver a continuación un ejemplo de este formulario básico construido con los elementos presentados anteriormente.

Figura 9.

Prueba de formulario

Nombre: Jose
Apellidos: Prueba
Clave: ****

Sexo:
Hombre Mujer
Aficiones:
Deportes Música Lectura
Procedencia:
África

Donde le gustaria viajar:
África
Antártida
América
Asia
Opine:
Escriba aquí su opinión!

Submit Query Reset

ANOTACIONES

El código HTML que ha producido este resultado es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>
<H1>Prueba de formulario</H1>

<FORM METHOD=GET>
Nombre: <INPUT TYPE=TEXT NAME=NOMBRE SIZE=10><BR>
Apellidos: <INPUT TYPE=TEXT NAME=APELLIDOS SIZE=30><BR>
Clave: <INPUT TYPE=PASSWORD NAME=PASS SIZE=8><BR>
<HR>
Sexo: <BR>
<INPUT TYPE="RADIO" NAME="SEXO">Hombre
<INPUT TYPE="RADIO" NAME="SEXO">Mujer
<BR>
Aficiones:<BR>
<INPUT TYPE="CHECKBOX" NAME="DEPORTES">Deportes
<INPUT TYPE="CHECKBOX" NAME="MUSICA">Música
<INPUT TYPE="CHECKBOX" NAME="LECTURA">Lectura
<BR>
Procedencia:<BR>
<SELECT name="PROCEDENCIA">
 <option> África
 <option> Antártida
 <option> América
 <option> Asia
 <option> Europa
 <option> Oceanía
</SELECT>
<HR>
Dónde le gustaría viajar:<BR>
<SELECT name="destino" MULTIPLE SIZE=4>
 <option> África
 <option> Antártida
 <option> América
 <option> Asia
 <option> Europa
 <option> Oceanía
</SELECT>
<BR>
Opine:
<BR>
<TEXTAREA COLS=25 ROWS=10 NAME="OPINA">
¡Escriba aquí su opinión!
</TEXTAREA>
<HR>
<INPUT TYPE=SUBMIT> <INPUT TYPE=RESET>
</FORM>

</BODY>
</HTML>
```

ANOTACIONES

3.2. HTML avanzado

3.2.1. Hojas de estilo

Las hojas de estilo son un mecanismo para separar el formato de representación y presentación del contenido. Eso se consigue asociando atributos de presentación a cada una de las etiquetas de HTML o a subclases de éstas.

Por ejemplo, si deseamos que todos los párrafos de nuestro documento (definidos por <P></P>) tengan el fondo rojo y el texto en amarillo usaremos la siguiente definición:

```
<STYLE TYPE="text/css">
 P {color: red; background:yellow;}
</STYLE>
```

Para indicar qué estilos debemos usar en una página, disponemos de la etiqueta STYLE, que nos permite especificarlos *in situ*, y de otra etiqueta LINK, que nos permite indicar un fichero externo que contendrá nuestros estilos.

La etiqueta STYLE tiene que estar en la cabecera de la página. Dispondremos de un parámetro TYPE que nos permite indicar qué sintaxis usaremos para definir los estilos, que en nuestro caso será text/css. La etiqueta LINK, para poder definir una hoja de estilos externa, tiene la siguiente apariencia:

```
<LINK REL="stylesheet" HREF="miweb.css" TYPE="text/css">
```

De hecho es muy recomendable el uso de la etiqueta LINK para la definición de las hojas de estilo asociadas a una página, ya que de ese modo facilitaremos el mantenimiento, al concentrarse en un único fichero todos los estilos de un sitio web en lugar de estar replicados en cada una de las páginas.

ANOTACIONES

Formato de las hojas de estilo

Como hemos podido ver en los ejemplos anteriores, el formato de las hojas de estilo es el siguiente:

```
<elemento> {<formato>}
```

Por ejemplo:

```
P {color: red; background:yellow;}
```

Cabe destacar que la sintaxis de CSS (las hojas de estilo) es sensible a mayúsculas y minúsculas.

Esta sintaxis nos permitiría definir el formato que deseamos para los párrafos de nuestra web. Existe una extensión de dicha sintaxis que hace posible definir un estilo que sólo se aplicará a ciertas partes de nuestro documento. En concreto, nos permite definir `clases` de elementos a los que aplicaremos el estilo.

Por ejemplo, para definir una clase de párrafo que llamaremos destacado:

```
P.destacado {color: red; background:yellow;}
```

Podemos usar luego el atributo `CLASS` que HTML 4.0 añadió a HTML para definir la clase de cada párrafo:

```
<P CLASS="destacado">Un párrafo destacado</P>
<P>Un párrafo normal</P>
<P CLASS="destacado">Otro destacado</P>
```

Existe además un método para asignar un estilo a párrafos individuales, aportando así mayor granularidad al concepto de clases. Para ello deberemos definir el estilo de un elemento individual de HTML mediante CSS usando la siguiente sintaxis:

```
#parrafo1 {color: green; background:yellow;}
```

ANOTACIONES

A continuación, podemos otorgar a algún elemento de HTML dicha identidad mediante el uso del atributo ID:

```
<P CLASS="destacado">Un párrafo destacado</P>
<P>Un párrafo normal</P>
<P CLASS="destacado" ID="parrafo1">Otro destacado pero
el color en éste viene dado por su identidad</P>
```

Las etiquetas SPAN y DIV

Hemos visto cómo asignar estilos a los elementos de HTML (párrafos, etc.), pero en algunas ocasiones es interesante asignar estilos a bloques de texto o de contenido que no formen parte de ningún bloque de HTML.

Ejemplo

Por ejemplo, podría interesarnos definir un estilo que permitiese marcar palabras concretas del texto (para indicar cambios, etc.). Evidentemente, no podemos definir una nueva etiqueta de HTML, ya que ello comportaría que los navegadores existentes que desconocen nuestra etiqueta ignorarían dicho contenido.

La solución viene de la mano de la etiqueta `DIV`, que ya hemos visto, y de `SPAN`.

Si queremos marcar un bloque de contenido como perteneciente a una clase concreta para poder definir un estilo para él o darle una identificación individual, deberemos envolver dicho contenido con una etiqueta `SPAN` o `DIV`. La diferencia entre usar una u otra consistirá en que `DIV` se asegura de que tanto al inicio del bloque como al final haya un salto de línea. Podemos, así, definir estilos para bloques de texto sin que tengamos que envolverlos con etiquetas que modificarían su formato (como P).

Por ejemplo, podemos definir:

```
all.dudoso
{
   color: red;
}
all.revisado
{
   color:blue;
}
```

ANOTACIONES

y usarlo luego en nuestro documento HTML:

```
<P>Este párrafo <SPAN CLASS="dudoso">largo</SPAN> deberá ser
corregido por el <SPAN CLASS="revisado">director general</SPAN>
</P>
```

Propiedades más importantes

Veremos algunas de las propiedades más importantes que podemos fijar gracias a CSS. Cabe destacar que, debido a las incompatibilidades existentes entre los diferentes navegadores, es recomendable probar nuestras páginas con diferentes versiones de navegador y con navegadores distintos a fin de asegurar su correcta visualización.

- **Propiedades relativas a tipos de letra**

Las propiedades que nos permitirán definir el aspecto (tipografía) del texto son:

– `font-family`: tipo de letra (que puede ser genérico entre: `serif`, `cursive`, `sans-serif`, `fantasy` y `monospace`). Podemos especificar no un sólo tipo, sino una lista de tipos, genéricos o no, separados por comas. Al especificar tipos, hay que recordar que éstos pueden no estar presentes en el ordenador del usuario que visite nuestra página.

– `font-size`: tamaño del tipo de letra. `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, además de valores numéricos de tamaño relativo o tamaño absoluto.

– `font-weight`: grosor del tipo de letra. Los valores posibles son: `normal`, `bold`, `bolder`, `lighter` y valores numéricos entre 100 y 900 (donde 900 será el tipo de negrilla más grueso).

– `font-style`: estilo del tipo de letra. Disponemos de `normal`, `italic`, `italic small-caps`, `oblique`, `oblique small-caps` y `small-caps`.

ANOTACIONES

- **Propiedades del texto**

Disponemos también de un grupo de propiedades que nos permiten alterar el texto de nuestra página y el formato de éste.

- `line-height`: interlineado en valor numérico o porcentual.
- `text-decoration`: *decoración* del texto: `none`, `underline` (subrayado), `overline` (sobrerrayado), `line-through` (tachado) o `blink` (parpadeo).
- `vertical-align`: alineación vertical del texto. Disponemos de: `baseline` (normal), `sub` (subíndice), `super` (superíndice), `top`, `text-top`, `middle`, `bottom`, `text-bottom` o un porcentaje.
- `text-transform`: modificaciones del texto: `capitalize` (la primera letra en mayúsculas), `uppercase` (convierte a mayúsculas el texto), `lowecase` (lo convierte a minúsculas) o `none`.
- `text-align`: alineación horizontal del texto: `left`, `right`, `center` o `justify`.
- `text-indent`: sangrado de la primera línea del texto en valor absoluto o porcentual.

- **Propiedades de bloques**

Las siguientes propiedades afectan a los bloques de texto (párrafos, etc.).

- `margin-top`, `margin-right`, `margin-bottom`, `margin-left`: distancia mínima entre un bloque y los elementos adyacentes. Valores posibles: tamaño, porcentaje o `auto`.
- `padding-top`, `padding-right`, `padding-bottom`, `padding-left`: relleno de espacio entre el borde y el contenido del bloque. Valores posibles: tamaño en valor absoluto, porcentual o `auto`.

ANOTACIONES

– border-top-width, border-right-width, border-bottom-width, border-left-width: anchura del borde del bloque en valor numérico.

– border-style: estilo del borde del bloque. none, solid o 3D.

– border-color: color del borde del bloque.

– width, height: medidas del bloque. Valores en porcentaje, valores absolutos o auto.

– float: justificación del contenido de un bloque. Valores: left, right o none.

– clear: colocación de los otros elementos respecto al actual. Valores posibles: left, right, both o none.

• Otras propiedades

Disponemos, además, de otras propiedades de las hojas de estilo que nos permiten modificar otros aspectos:

– color: color del texto.

– background: color de fondo o imagen de fondo. Valores, un color o una URL del fichero de imagen.

```
background: url(fondobonito.gif);
```

– display: decide el carácter de bloque o no de un elemento. Puede ser: inline (como <I> o <B>), block (como <P>, list como <LI> o none, que inhabilita el elemento).

– list-style: estilo de marcador de un elemento de una lista (podemos así usar un gráfico como marcador). Valores posibles: disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none o una URL de una imagen.

– white-space: indica qué tratamiento deben tener los espacios en blanco, si el habitual o deben ser respetados tal cual, como en el bloque <PRE>. Valores: normal y pre.

Nota

En CSS el formato para una URL es el siguiente: url (dirección).

ANOTACIONES

3.2.2. Capas

HTML 4.0 introdujo un nuevo concepto que permite un mayor control en el posicionamiento de los elementos en nuestras páginas. Podemos definir las capas como páginas incrustadas dentro de otras páginas.

Es posible especificar los atributos de las capas (posición, visibilidad, etc.) mediante hojas de estilo, al igual que los de los demás elementos de HTML. Las capas, con la posibilidad que ofrecen de poder controlarse desde lenguajes de programación como JavaScript, son la base de lo que conocemos como HTML dinámico. Desgraciadamente, las implementaciones de los diversos navegadores son incompatibles entre sí, por lo que nos veremos obligados bien a usar cantidades ingentes de código de programa para prever todas las posibilidades o bien a limitarnos a usar sólo los mínimos comunes. Una de las pocas opciones que tenemos para hacer que funcionen las capas en la mayoría de navegadores estriba en definir las capas mediante hojas de estilo CSS.

Podemos ver en el ejemplo siguiente cómo colocar una capa que identificaremos como `lacapa` mediante el uso de `ID`.

```
<STYLE TYPE="text/css">
   #lacapa {position:absolute; top:50px; left:50px;}
</STYLE>
```

Este ejemplo colocaría la capa `lacapa` a 50 puntos de la esquina superior izquierda de la página. En este caso, para definir la capa usaremos una etiqueta de tipo `SPAN`:

```
<SPAN ID="lacapa">
   ...
   Contenido de la capa
   ...
</SPAN>
```

Como hemos visto, hemos colocado la capa anterior en una posición concreta dentro de la página. De igual modo, podemos colocar las capas en posiciones relativas respecto a la posición que ocuparía el texto en la página donde están escritas.

ANOTACIONES

La definición es la siguiente:

```
<STYLE TYPE="text/css">
   #capaflot {position: relative; left: 20px; top: 100px;}
</STYLE>
```

Disponemos de diversas propiedades específicas de las capas que podemos modificar convenientemente:

- `left`, `top`: nos permitirán indicar la posición de la esquina superior izquierda de la capa respecto a la capa donde esté situada. El documento completo se considera una capa.
- `height`, `width`: indicarán la altura y anchura de la capa.
- `clip`: nos permite definir un área de recorte en el interior de la capa.
- `z-index`: indica la profundidad en la pila de capas. A mayor `z-index`, menor profundidad y mayor visibilidad (se superpondrán a las de menor `z-index`). Por defecto, el `z-index` se asigna por orden de definición en el fichero HTML.
- `visibility`: especifican si la capa debe ser visible u oculta. Los valores posibles son `visible`, `hidden` (oculta) o `inherit` (hereda la visibilidad de la capa padre).
- `background-image`: imagen que formará el fondo de la capa.
- `background-color`, `layer-background-color`: definen el color de fondo de la capa para Internet Explorer y Mozilla/Netscape, respectivamente.

3.3. HTML dinámico

El HTML dinámico (DHTML o *dynamic* HTML) no es un estándar definido por el W3C, sino que es un término de márqueting que utilizan Netscape y Microsoft para referirse al conjunto de nuevas tecnologías de web. Dicho conjunto comprende:

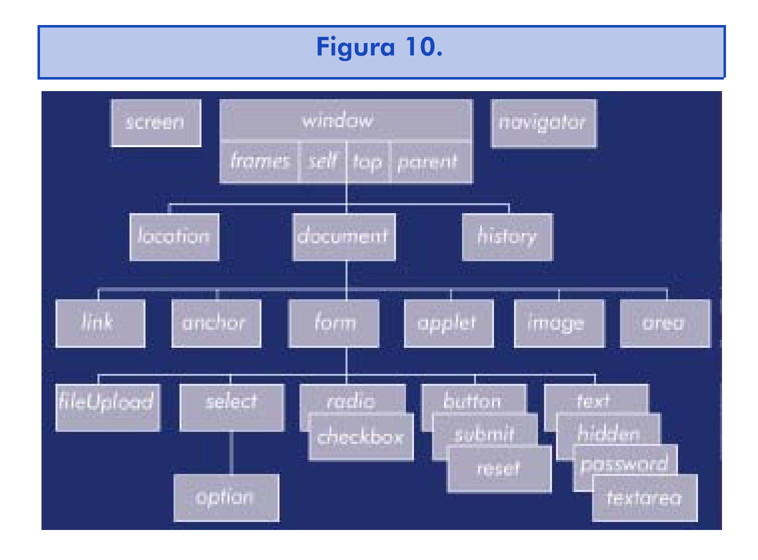
- HTML, especialmente HTML 4.0
- Hojas de estilo (CSS)
- Javascript

ANOTACIONES

Se suele calificar como DHTML a este conjunto de tecnologías, especialmente en aquellos casos en que operan conjuntamente para enriquecer la experiencia web del usuario. Esta conjunción de tecnologías permite, entre otras cosas, ofrecer al usuario interfaces gráficas mucho más ricas y complejas, controlar formularios de forma más eficiente (el código Javascript se ejecuta en el cliente, con el consiguiente incremento de rendimiento), etc.

Uno de los puntos claves de DHTML es DOM *(document object model*), que define una jerarquía de objetos accesibles mediante Javascript (un árbol de hecho) que representan todos y cada uno de los elementos del documento HTML. El árbol usado en DOM es el siguiente:

Figura 10.



Veremos un ejemplo de cómo podemos definir un formulario en HTML que permite, mediante controles dirigidos por Javascript y DOM, manipular un elemento de tipo TEXTAREA.

```
<html>
 <head>
  <meta http-equiv="Content-Type"
   content="text/html; charset=ISO-8859-1">
   <title>Textarea</title>.
   <meta name="Author" content="ShinSoft">
   <meta http-equiv="Content-Script-Type" content="text/javascript">
   <meta http-equiv="Content-Style-Type" content="text/css">
   <style type="text/css">
   <!--
table{ background-color:#99ff99; border:2px solid #66cc66; }
```

ANOTACIONES

```
td textarea                                    { background-color:#ffffff;
  border:2px inset #66cc66; width:100 %; }
td input[type="button"]                        { background-color:#ccccff;
  border:2px outset #9999ff; }
td input[type="text"]                          { background-color:#ffffee;
  border:2px solid #ff9999; text-align:right; }
[readonly]{ color:#999966; }
dt { font-weight:bold; font-family:fantasy; }
#t { background-color:#ffffee; border:2px solid #ff9999; }
     -->
     </style>

<script language="JavaScript">
<!--
function notSupported(){ alert('El navegador no lo soporta.'); }
function setSel(){
 var f=document.f;
 var t=f.ta;
if(t.setSelectionRange){
  var start=parseInt(f.start.value);
  var end =parseInt(f.end .value);
  t.setSelectionRange(start,end);
  t.focus();
  f.t.value = t.value.substr(t.selectionStart,
 t.selectionEnd-t.selectionStart);
 } else notSupported();
}
function setProp(id){
 var f=document.f;
 var t=f.ta;
 if(id==0) t.selectionStart = parseInt(f.start.value);
 if(id==1) t.selectionEnd = parseInt(f.end .value);
 f.t.value = t.value.substr(t.selectionStart,
   t.selectionEnd-t.selectionStart);
 t.focus();
}
function getProp(id){
 var f=document.f;
 var t=f.ta;
 if(id==0) f.start.value = t.selectionStart;
 if(id==1) f.end.value = t.selectionEnd;
 if(id==2) f.txl.value = t.textLength;
 f.t.value = t.value.substr(t.selectionStart,
   t.selectionEnd-t.selectionStart);
 t.focus();
}
function init(){
 var f=document.f;
 var t=f.ta;
```

ANOTACIONES

```
if(t.setSelectionRange){
 f.start.value = t.selectionStart
 f.end .value = t.selectionEnd;
 f.txl .value = t.textLength;
 } else notSupported();
}
// -->
</script>
</head>
<body bgcolor="#ffffff" text="#000000"
     link="#cc6666" alink="#ff0000" vlink="cc6666"
        onLoad="init();">
<h2>Elemento Textarea</h2>
<form name="f">
<table border=0 cellspacing=1>
     <tr>
     <th>Inicio de la selección</th>
     <td>
     <input type="text" name="start" size=4 value="0">.
          <input type="button" value="obtener" onClick="getProp(0);">
         <input type="button" value="poner" onClick="setProp(0);">
     </td>
     <td rowspan=2>
     <input type="button" value="Seleccionar" onClick="setSel();">
     </td>
</tr>
<tr>
   <th>Final de la selección</th>
     <td>
     <input type="text" name="end" size=4 value="1">
          <input type="button" value="obtener" onClick="getProp(1);">
          <input type="button" value="poner" onClick="setProp(1);">
     </td>
     </tr>
<tr>
     <th>Longitud del texto</th>
     <td>
     <input type="text" name="txl" id="txl" size=4 value="" readonly>
           <input type="button" value="obtener" onClick="getProp(2);">
     </td>
 </tr>
<tr><th>Elemento TextArea</th><td colspan=2>
<textarea name="ta" id="ta" cols=30 rows=5>
Podemos seleccionar partes de este texto
</textarea></td></tr>
<tr>
     <th>selected string</th>
     <td colspan=2>
     <textarea name="t" id="t" readonly></textarea>
     </td>
```

ANOTACIONES

```
</tr>
</table>
</form>

<dl>
<dt>Botón poner:</dt>
<dd>Asignar el valor en función del contenido de Textarea.</dd>
<dt>Botón Seleccionar:</dt>

<dd>Utiliza los valores para seleccionar texto.</dd>
<dt>Botón obtener:</dt>
<dd>Obtener los valores en función de lo seleccionado.</dd>
</dl>

</body></html>
```

El resultado que aparece en un navegador es el siguiente:

Figura 11.

Elemento Textarea

Inicio de la seleccion	41	obtener	poner	Seleccionar
Final de la seleccion	41	obtener	poner	
Longitud del texto	41	obtener		
Elemento TextArea	Podemos seleccionar partes de este texto			
selected string				

Botón poner:
Asignar el valor en función del contenido de Textarea.
Botón Seleccionar:
Utiliza los valores para seleccionar texto.
Botón obtener:
Obtener los valores en función de lo seleccionado.

3.4. Javascript

Javascript es un lenguaje de programación interpretado (un lenguaje de tipo *script*). A pesar de que existen intérpretes no dependientes de ningún navegador, es un lenguaje de *script* que suele encontrarse

vinculado a páginas web. Javascript y Java son dos lenguajes de programación distintos con filosofías muy diferentes. El único punto en común es la sintaxis, ya que cuando Netscape diseñó Javascript, se inspiró en la sintaxis de Java.

3.4.1. El primer programa sencillo

Como resulta habitual al mostrar lenguajes de programación, nuestro primer contacto con Javascript será realizar un primer programa que nos muestre el ya clásico mensaje "Hola mundo". Como Javascript es un lenguaje que veremos, generalmente, vinculado a una página web, el siguiente código será un fichero HTML que deberemos visualizar en un navegador.

```
<HTML>
 <HEAD>
    <SCRIPT LANGUAGE="Javascript">
       function Saludo()
       {
          alert("Hola mundo");
       }
    </SCRIPT>
 </HEAD>
 <BODY>
   <FORM>
      <INPUT TYPE="button" NAME="Boton"
             VALUE="Pulsar" onClick="Saludo()">
   </FORM>
 </BODY>
</HTML>
```

Este programa Javascript pinta en nuestra pantalla un botón. Al pulsarlo, se abre una ventana con un mensaje. El aspecto que ofrece este programa es el siguiente:

Figura 12.



ANOTACIONES

Comentaremos el código anterior, lo cual nos permitirá introducir los diversos elementos de Javascript.

Como podemos ver, el código Javascript se encuentra envuelto por una etiqueta <SCRIPT>. Dicha etiqueta puede aparecer en el punto del documento que deseemos (no es obligatorio que aparezca en la cabecera de éste). Los navegadores que no ofrezcan soporte de Javascript ignorarán el contenido de la etiqueta. Podemos, opcionalmente, requerir una versión concreta de Javascript si usamos una etiqueta como:

```
<SCRIPT LANGUAGE="Javascript1.1">
...
</SCRIPT>
```

Una forma conveniente de usar la etiqueta SCRIPT es situarla en el encabezado de la página, ya que así se mejora la legibilidad del código HTML.

En el interior de la etiqueta SCRIPT tenemos el código Javascript. En este caso, una única función, aunque podríamos tener más.

Nuestro código:

```
function Saludo()
{
   alert("Hola mundo");
}
```

define una función llamada Saludo. Como podemos observar, y a diferencia de Java, esta función no pertenece a ningún objeto. Javascript, a pesar de ser orientado a objetos, permite la existencia de funciones fuera de los objetos (de forma similar a C++). Veremos que el único código que contiene esta función es una llamada a función, alert (un método del objeto window).

El siguiente bloque de código Javascript está dentro de la definición en HTML del formulario.

```
<FORM>
   <INPUT TYPE="button" NAME="Boton"
   VALUE="Pulsar" onClick="Saludo()">
</FORM>.
```

ANOTACIONES

En este caso, el código Javascript declara un gestor de eventos, concretamente del evento `onClick` para el objeto botón. Un evento es una ocurrencia de algún hecho (en este caso, un click del botón por parte del usuario). Al producirse el evento Javascript se ejecutará el código que le indicamos en el gestor de eventos `onClick`. Este código es una llamada a una función, `Saludo`.

3.4.2. Elementos básicos de Javascript

Las sentencias de Javascript terminan en `;` (igual que en C y Java) y se pueden agrupar por bloques delimitados por { y }.

Otro punto que conviene tener presente es que los símbolos (nombres de variables, funciones, etc.) son sensibles a cambios de mayúsculas/minúsculas.

Comentarios

Para introducir comentarios en el programa disponemos de dos opciones:

```
// Comentario de una sola línea

/*
   Comentario que se extiende
   varias líneas
*/
```

Como podemos apreciar, el formato de los comentarios es idéntico al de Java.

Literales

Javascript sigue el mecanismo de definición de literales de Java y C. Es decir, disponemos de literales de tipo:

- **Enteros** 123
- **Reales** 0.034
- **Booleanos** true, false
- **Cadenas** "Cadena de texto"

ANOTACIONES

Javascript también proporciona soporte para vectores:

```
estaciones= ["Otoño","Invierno","Primavera","Verano"];.
```

Caracteres especiales

Javascript, al igual que Java, utiliza ciertas secuencias de caracteres para permitirnos introducir caracteres especiales en nuestras constantes de cadena.

Dentro de las cadenas, podemos indicar varios caracteres especiales, con significados especiales. Éstos son los más usados:

Tabla 3. Caracteres

Carácter	Significado
`\n`	Nueva línea
`\t`	Tabulador
`\'`	Comilla simple
`\"`	Comilla doble
`\\`	Barra invertida
`\xxx`	El número ASCII (según la codificación Latin-1) del carácter en hexadecimal

3.4.3. Tipos de datos y variables

En Javascript los tipos de datos se asignan dinámicamente a medida que asignamos valores a las distintas variables y pueden ser:

- cadenas de caracteres
- enteros
- reales
- booleanos
- vectores
- matrices
- referencias
- objetos

ANOTACIONES

ANOTACIONES

Variables

En Javascript los nombres de variables deben empezar por un carácter alfabético o por el carácter '_ ' y pueden estar formados por caracteres alfanuméricos y el carácter '_ '.

No hace falta declaración explícita de las variables, ya que éstas son globales. Si deseamos una variable local, debemos declararla usando la palabra reservada `var` y hacerlo en el cuerpo de una función. En una declaración de variable mediante `var` podemos declarar diversas variables, separando los nombres de éstas mediante ,.

Las variables tomarán el tipo de datos a partir del tipo del objeto de datos que les asignemos.

Referencias

Javascript elimina del lenguaje los punteros a memoria, pero mantiene el uso de referencias. Una referencia funciona de forma muy similar a un puntero de memoria, pero obviando para el programador las tareas relacionadas con gestión de memoria que provocaban tantos errores en los punteros dentro de otros lenguajes.

En Javascript se dispone de referencias a objetos y a funciones. Esta capacidad de tener referencias a funciones nos será muy útil para utilizar funciones que oculten las diferencias entre navegadores.

```
function soloExplorer()
{
  ...
}
function soloMozilla()
{
  ...
}

function toda()
{
   var funcion;
   if(navegadorMozilla)
      funcion=soloMozilla;
   else
      funcion=soloExplorer;

   funcion();
}
```

Vectores

Disponemos en Javascript de un tipo de datos que nos permite manejar colecciones de datos. Los elementos de estos `Array` pueden ser distintos.

Como podemos ver en el código siguiente, los `Array` de Javascript son objetos (de tipo `Array`) los cuales pueden tener como índice de acceso un valor no numérico y de los que no debemos declarar las medidas inicialmente. No disponemos de un tipo de vector *n*-dimensional, pudiendo usar para ello vectores de vectores.

```
//dimensionamos un vector de 20 elementos
vector = new Array(20);

//el vector crece para albergar el 30 elemento
miEstupendoVector[30] = "contenido";.

//dimensionamos un vector
capitales = new Array();

//podemos usar cadenas como indices
capitales["Francia"]= "París";
```

Operadores

Disponemos en Javascript de los mismos operadores que en Java y C, siendo su comportamiento el habitual de estos lenguajes:

- **Operadores aritméticos**: disponemos de los habituales operadores aritméticos (+, -, *, /, %, etc.), además de los operadores de incremento (++) y decremento (–).

- **Operadores de comparación**: disponemos de los siguientes:

  – Igualdad ==
  – Desigualdad !=
  – Igualdad estricta ===
  – Desigualdad estricta !==
  – Menor que <

ANOTACIONES

– Mayor que >
– Menor o igual que <=
– Mayor o igual que >=

• **Operadores lógicos:** Javascript proporciona los siguientes operadores lógicos:

– Negación `!`
– Y `&&`
– Ó `||`

• **Operadores de objeto:** además, para la manipulación de objetos disponemos de:

– Crear un objeto `new`
– Borrar un objeto `delete`
– Referencia al objeto actual `this`

3.4.4. Estructuras de control

Javascript, como todos los lenguajes de programación, proporciona algunas estructuras de control.

Bifurcaciones condicionales

Javascript proporciona las dos estructuras de control más conocidas:

```
if (condicion)
   <codigo>
else
   <codigo>
switch(valor)
 {
   case valortest1:
      <codigo>
      break;
   case valortest2:
      <codigo>
      break;
   ...
   default:
      <codigo>
 }
```

ANOTACIONES

Bucles

Dispondremos de tres bucles, el `while` y `do, while` y el bucle `for`.

```
while(condicion)
 <codigo>

do
 {
 <codigo>
 } while(condicion);

for(inicio; condicion; incremento)
 <codigo>
```

Estructuras de manejo de objetos

Disponemos de dos estructuras muy peculiares para manejo de objetos. Por un lado, tenemos el bucle `for..in` que permite recorridos por las propiedades de un objeto (generalmente en vectores):

```
for (<variable> in <objeto)
 <codigo>
```

Por otro lado tenemos `with`, que proporcionará una mayor comodidad cuando tengamos que tratar con múltiples propiedades de un mismo objeto. Podemos escribir:

```
with (objeto)
{
 propiedad1 = ...
 propiedad2 = ...
}
```

en lugar de:

```
objeto.propiedad1=...
objeto.propiedad2=...
```

3.4.5. Funciones

Javascript proporciona las construcciones necesarias para definir funciones. La sintaxis es la siguiente:

```
function nombre(argumento1, argumento2,..., argumento n)
{
 código
}
```

Los parámetros se pasan por valor.

3.4.6. Objetos

Un objeto en Javascript es una estructura de datos que contiene tanto variables (propiedades del objeto), como funciones que manipulan el objeto (métodos). El modelo de programación orientada a objetos de Javascript es mucho más simple que el de Java o C++. En Javascript no se distingue entre objetos e instancias de objetos.

El mecanismo para acceder a las propiedades o a los métodos de un objeto es el siguiente:

```
objeto.propiedad
valor=objeto.metodo(parametro1, parametro2, ...)
```

Definición de objetos en Javascript

Para definir un objeto en Javascript debemos, en primer lugar, definir una función especial, cuya finalidad consiste en construir el objeto. A dicha función, llamada constructor, debemos asignarle el mismo nombre que al objeto.

```
function MiObjeto(atr1, atr2)
{
 this.atr1=atr1;
 this.atr2=atr2;
}
```

ANOTACIONES

A partir de este momento podemos crear objetos de tipo `MiObjeto`.

```
objeto=new MiObjeto(....)
objeto.atr1=a;.
```

Para añadir métodos a un objeto, debemos definir primero dichos métodos como una función normal:

```
function Metodo1(atr1, atr2)
{
 //codigo
 // en this tenemos el objeto
}
```

Para asignar este método a un método del objeto escribiremos:

```
objeto.metodo1=Metodo1;
```

A partir de ese momento podemos realizar:

```
objeto.metodo1(....);
```

Herencia

La herencia en la programación orientada a objetos nos permite crear objetos nuevos que dispongan de los métodos y propiedades de unos objetos a los que llamaremos *padre*. Gracias a ello, podemos crear objetos derivados y pasar de implementaciones genéricas a implementaciones cada vez más concretas.

La sintaxis para crear un objeto derivado de otro, por ejemplo, un objeto `ObjetoHijo` derivado de un objeto `ObjetoPadre`, el cual tenía dos argumentos (`arg1` y `arg2`), será:

```
function ObjetoHijo(arg1, arg2, arg3)
{
 this.base=ObjetoPadre;
 this.base(arg1,arg2);

}
```

En ese momento podemos acceder por un objeto de tipo `ObjetoHijo` tanto a los métodos y propiedades del objeto hijo como del objeto padre.

Objetos predefinidos

En las implementaciones de Javascript existentes disponemos de un conjunto de objetos ya predefinidos:

- **Array** Vectores.
- **Date** Para almacenar y manipular fechas.
- **Math** Métodos matemáticos y constantes.
- **Number** Algunas constantes.
- **String** Manejo de cadenas.
- **RegExp** Manejo de expresiones regulares.
- **Boolean** Valores booleanos.
- **Function** Funciones.

3.4.7. Eventos

Uno de los puntos más importantes de Javascript es su interacción con el navegador. Para ello incorpora una serie de eventos que se disparan en el momento en que el usuario realiza alguna acción en la página web.

Tabla 4. Eventos

Evento	Descripción
onLoad	Termina de cargarse una página. Disponible en: <BODY>
onUnLoad	Salir de una página. Disponible en: <BODY>
onMouseOver	Pasar el ratón por encima. Disponible en: <A>, <AREA>
onMouseOut	Que el ratón deje de estar encima de un elemento
onSubmit	Enviar un formulario. Disponible en: <FORM>

ANOTACIONES

Evento	Descripción
onClick	Pulsar un elemento. Disponible en: <INPUT>
onBlur	Perder el cursor. Disponible en: <INPUT>, <TEXTAREA>
onChange	Cambiar de contenido. Disponible en: <INPUT>, <TEXTAREA>
onFocus	Conseguir el cursor. Disponible en: <INPUT>, <TEXTAREA>
onSelect	Seleccionar texto. Disponible en: <INPUT TYPE="text">, <TEXTAREA>

Disponemos de dos mecanismos para indicar qué función manejará un evento:

1.

```
<HTML>
<HEAD>
 <SCRIPT LANGUAGE="Javascript">
    function Alarma() {
       alert("Hola Mundo");
    }
 </SCRIPT>
</HEAD>
<BODY onLoad="Saludo()">
...
</BODY>
</HTML>
```

2.

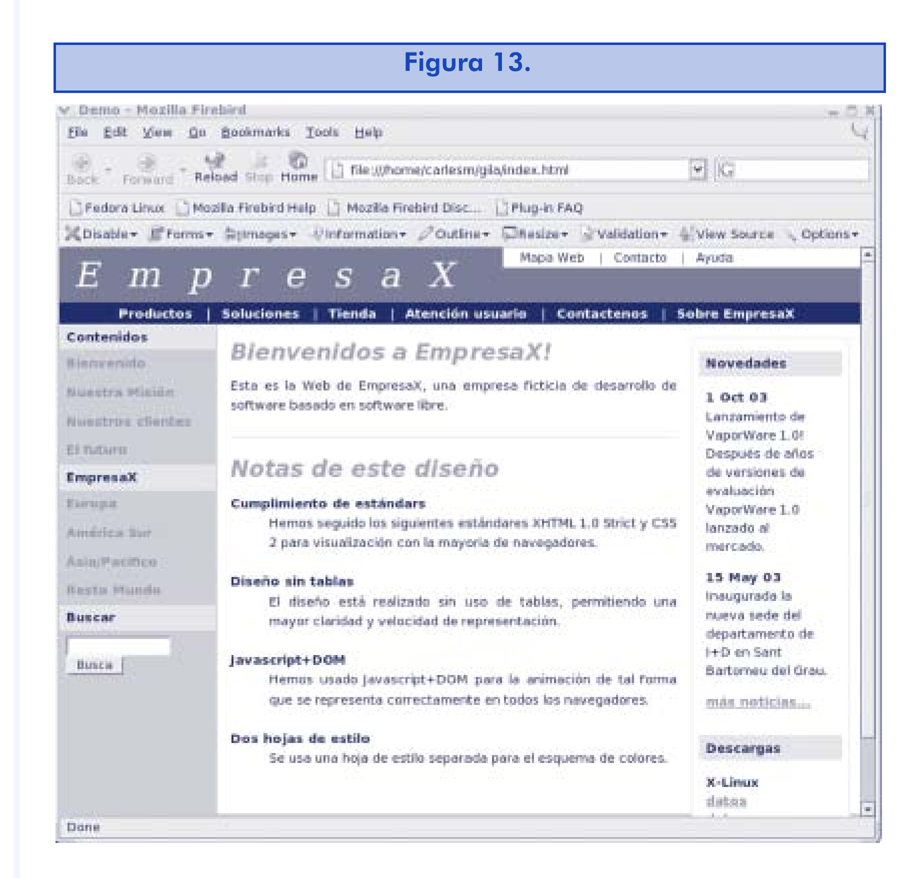
```
<HTML>
<HEAD>
 <SCRIPT LANGUAGE="Javascript">
    function Saludo() {
       alert("Hola Mundo");
    }
    window.onload = Saludo;.
 </SCRIPT>
</HEAD>
<BODY>
...
</BODY>
</HTML>.
```

ANOTACIONES

3.5. Prácticas: creación de una página web compleja con las técnicas presentadas

Vamos a crear ahora una página web usando todas las técnicas presentadas hasta el momento. El resultado que obtendremos será una página como la que sigue:

Figura 13.



Para construir esta página no vamos a usar en ningún momento tablas o marcos (*frames*). Vamos a usar únicamente separadores `DIV`, `P`, etc., así como posicionamiento por CSS.

Incluiremos, además, un efecto de animación con el título de la empresa para el que no utilizaremos ningún tipo de componente que no sea el propio DHTML, lo cual nos permitirá diseñar dicha animación de tal forma que funcione correctamente tanto en navegadores Mozilla/Netscape u Opera, como Internet Explorer, etc.

Otro de los principios del diseño de dicha página reside en que vamos a usar dos ficheros diferentes para mantener las hojas de estilo: uno con los colores y el otro con los estilos propiamente dichos. Ello nos facilitará posibles cambios de estilo, etc.

Mostraremos a continuación la página de antes, indicando ahora el tipo de elemento en cada uno de los bloques que constituyen la página:

Figura 14.



Éste es el código correspondiente a la hoja de estilos que nos indica el formato, denominada en nuestro ejemplo `estilo.css`

```
/* ###### Body ###### */

body {
 font-family: verdana, tahoma, helvetica, arial, sans-serif;
 font-size: 94 %;
 margin: 0;
}

h1, h2, h3 {
   font-family: arial, verdana, tahoma, sans-serif;
}
```

```
h1 {
 font-size: 164 %;
 font-weight: bold;
 font-style: italic;
 padding-top: 1em;
 border-top-style: solid;
 border-top-width: 1px;
}

p {
 padding-bottom: 1ex;
}
img {
 border: none;
}

code {
 font-family: "lucida console", monospace;
 font-size: 95 %;
}
dt {
 font-weight: bold;
}
dd {
 padding-bottom: 1.5em;
}
#textoCuerpo {
 text-align: justify;
 line-height: 1.5em;
 margin-left: 12em;
 padding: 0.5ex 14em 1em 1em;
 border-left-style: solid;
 border-left-width: 1px;
}

#textoCuerpo a {
 /* colores.css */
}
#textoCuerpo a:hover {
 text-decoration: none;
}
```

ANOTACIONES

```
/* ###### Cabecera ###### */

#cabecera{
 height: 4em;
 padding: 0.25em 2.5mm 0 4mm;
}

.cabeceraTitulo {
 font-size: 252 %;
 text-decoration: none;
 font-weight: bold;
 font-style: italic;
 line-height: 1.5em;
}

.cabeceraTitulo span {
 font-weight: normal;
}

.cabeceraEnlaces {
 font-size: 87 %;
 padding: 0.5ex 10em 0.5ex 1em;
 position: absolute;
 right: 0;
 top: 0;.
}

.cabeceraEnlaces * {
 text-decoration: none;
 padding: 0 2ex 0 1ex;
}

.cabeceraEnlaces a:hover {
 text-decoration: underline;
}

.barraMenu {
 text-align: center;
 padding: 0.5ex 0;
}
```

ANOTACIONES

```
.barraMenu * {
 text-decoration: none;
 font-weight: bold;
 padding: 0 2ex 0 1ex;
}

.barraMenu a:hover {
 /* colores.css */
}

/* ###### Izquierda ###### */
.barraIzquierda {
 font-size: 95 %;
 width: 12.65em;
 float: left;
 clear: left;
}

.barraIzquierda a, .barraIzquierda span {
 text-decoration: none;
 font-weight: bold;
 line-height: 2em;
 padding: 0.75ex 1ex;
 display: block;
}
[class˜="barraIzquierda"] a, [class˜="barraIzquierda"] span {
 line-height: 1.5em;
}

.barraIzquierda a:hover {
 /* colores.css */
}

.barraIzquierda .barraIzquierdaTitulo {
 font-weight: bold;
 padding: 0.75ex 1ex;
}

.barraIzquierda .barraTexto {
 font-weight: normal;
```

ANOTACIONES

```
 padding: 1ex 0.75ex 1ex 1ex;
}

.barraIzquierda .laPagina{
 /* colores.css */
}.

/* ###### Derecha ###### */

.barraDerecha {
 font-size: 95 %;
 width: 12.65em;
 margin: 2ex 0.8ex 0 0;
 float: right;
 clear: right;
 border-style: solid;
 border-width: 1px;
}
[class~="barraDerecha"] {
 margin-right: 1.5ex;
}

.barraDerecha a {
 font-weight: bold;
}
.barraDerecha a:hover {
text-decoration: none;
}

.barraDerecha .barraIzquierdaTitulo {
 font-weight: bold;
 margin: 1em 1ex;
 padding: 0.75ex 1ex;
}

.barraDerecha .barraTexto {
 font-weight: normal;
 line-height: 1.5em;
 padding: 0 3ex 1em 2ex;
}
```

ANOTACIONES

Mostraremos ahora la denominada colores.css:

```
/* ###### Colores del texto ###### */

#textoCuerpo a, .barraDerecha a
 { color: #ff2020; }

.barraIzquierda a
 { color: #ff2020; }

h1, .barraIzquierda span
 { color: #a68c53; }

.cabeceraTitulo, .cabeceraEnlaces *, .barraIzquierda
.barraIzquierdaTitulo, .barraIzquierda .laPagina,
.barraDerecha
.barraIzquierdaTitulo
{ color: black; }

.barraMenu a:hover, .barraIzquierda a:hover
{ color: black; }

.barraIzquierda a:hover
{ color: black; }

.cabeceraTitulo span, .barraMenu, .barraMenu *
{ color: white; }
.cabeceraEnlaces
 { color: #b82619; }

/* ###### Colores de fondo ###### */

body
{ background-color: #c3c2c2; }

#textoCuerpo, .cabeceraEnlaces, .barraMenu a:hover,
.barraDerecha
 { background-color: white; }

#cabecera
{ background-color: #b82619; }

.barraMenu
 { background-color: black; }

.barraIzquierda .barraIzquierdaTitulo, .barraDerecha
.barraIzquierdaTitulo
 { background-color: #e6dfcf; }

/* ###### Colores del borde ###### */

h1, #textoCuerpo, .barraDerecha
 { border-color: #e6dfcf; }
```

ANOTACIONES

Finalmente mostraremos el código correspondiente a la página web de *Empresa* X, nuestra empresa ficticia:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
  <head>
   <meta http-equiv="content-type" content="application/xhtml+xml; charset=iso8859-1" />
   <meta name="author" content="haran" />
   <meta name="generator" content="Windows Notepad" />
   <link rel="stylesheet" type="text/css" href="estilo.css" />
   <link rel="stylesheet" type="text/css" href="colores.css"/>

   <title>Demo</title>
<style type="text/css"> .textoexp {
 font-family: Arial;
color: black;
 font-size: 30pt;
 text-align: left;
 letter-spacing: -20px;
} </style> <script type="text/javascript"> function
expandirTexto(inicio, fin, paso, velo){
  if (inicio < fin){
  document.getElementById("textoexpand").style.letterSpacing = inicio+"px";
   inicio=inicio+paso;
   setTimeout("expandirTexto("+inicio+","+fin+","+paso+","+velo+")",velo);
  }
}

</script>
  </head>

   <body onLoad="expandirTexto(-20,30,1,5);">
   <div id="arriba"></div>

   <!-- ###### Cabecera ###### -->
   <div id="cabecera">
     <a href="./index.html" class="cabeceraTitulo" title="Homepage">
     <span class="textoexp" id="textoexpand">EmpresaX</span></a>
     <div class="cabeceraEnlaces">.
     <a href="./index.html">Mapa Web</a>
     <a href="./index.html">Contacto</a>
     <a href="./index.html">Ayuda</a>
   </div>
  </div>

  <div class="barraMenu">
   <a href="./index.html">Productos</a>
   <a href="./index.html">Soluciones</a>
   <a href="./index.html">Tienda</a>
   <a href="./index.html">Atenci&oacute;n usuario</a>
   <a href="./index.html">Contactenos</a>
   <a href="./index.html">Sobre EmpresaX</a>
  </div>
```

```
<!-- ###### Izquierda ###### -->

<div class="barraIzquierda">
 <div class="barraIzquierdaTitulo">Contenidos</div>
 <a href="#bien">Bienvenido</a>
 <a href="#mision">Nuestra Misión</a>
 <a href="#logros">Nuestros clientes</a>
<a href="#futuro">El futuro</a>
<div class="barraIzquierdaTitulo">EmpresaX</div>
 <a href="index.html">Europa</a>
 <a href="index.html">América Sur</a>
 <a href="index.html">ásia/Pacifico</a>
 <a href="index.html">Resto Mundo</a>

 <div class="barraIzquierdaTitulo">Buscar</div>
 <span class="barraTexto"><form method="GET">
 <Input type=text size=13 name="texto"><input type=submit
 name="Busca" value="Busca"></form></span>

</div>

<!-- ###### Derecha ###### -->

<div class="barraDerecha">
 <div class="barraIzquierdaTitulo">Novedades</div>
 <div class="barraTexto"><strong>1 Oct 03</strong><br />
 Lanzamiento de VaporWare 1.0! Después de años de versiones
 de evaluación VaporWare 1.0 lanzado al mercado.
 </div>
 <div class="barraTexto"><strong>15 May 03</strong><br />
   Inaugurada la nueva sede del departamento de I+D en
   Sant Bartomeu del Grau. </div>

 <div class="barraTexto"><a href="./index.html">más noticias...</a></div>
 <div class="barraIzquierdaTitulo">Descargas</div>
 <div class="barraTexto"><strong>X-Linux</strong><br />
   <a href="./index.html">datos </a>   <a href="./index.html">descarga</a></div>
 <div class="barraTexto"><strong>VaporWare</strong><br />
   <a href="./index.html">datos</a>   <a href="./index.html">descarga</a></div>

</div>

<!-- ###### Texto ###### -->
<div id="textoCuerpo">
 <h1 id="Bienvenida"
 style="border-top: none; padding-top: 0;">Bienvenidos a EmpresaX!</h1>
 <p>Esta es la Web de EmpresaX, una empresa ficticia de desarrollo de
 software basado en software libre.
 <h1 id="Notas">Notas de este diseño</h1>
 <dl>
  <dt>Cumplimiento de estándares</dt>
  <dd>Hemos seguido los siguientes estándares
    XHTML 1.0 Strict y CSS 2 para visualización con la mayoría
de navegadores.</dd>
```

```
<dt>Diseño sin tablas</dt>
    <dd>El diseño está realizado sin uso de tablas, permitiendo una mayor
      claridad y velocidad de representación.</dd>
    <dt>Javascript+DOM</dt>
    <dd>Hemos usado Javascript+DOM para la animación de tal forma que se
      representa correctamente en todos los navegadores.</dd>
    <dt>Dos hojas de estilo</dt>
    <dd>Se usa una hoja de estilo separada para el esquema de colores.</dd>
   </dl>
  </div>
 </body>
</html>
```

ANOTACIONES

4. Formato estructurado de texto: XML

4.1. Introducción a XML

XML son las siglas de eXtensible Markup Language, lenguaje extensible de marcas. Se trata de un estándar del Word Wide Web Consortium (`http://www.w3.org` es una referencia básica de XML), cuyo objetivo original consistía en permitir afrontar los retos de la publicación electrónica de documentos a gran escala. Actualmente, XML está empezando a desempeñar un papel muy importante en el intercambio de una gran variedad de información en la web y en otros contextos.

XML deriva del lenguaje de marcas SGML (un estándar ISO, concretamente el ISO-8879). Concretamente, es un subconjunto de SGML que pretende que éste pueda ser servido, recibido y procesado en la web de la misma forma que el HTML. XML ha sido diseñado buscando la simplicidad de implementación y la interoperabilidad con SGML y HTML y tratando, por otra parte, de que sea usado para diseñar aplicaciones centradas en los datos.

XML nace en 1996, desarrollado por un grupo auspiciado por el W3C y al que inicialmente se conocía como grupo de trabajo de SGML, con los siguientes objetivos, tal como se enumeran en el documento de definición del estándar:



1. XML debe ser utilizable directamente sobre Internet.
2. XML debe soportar una amplia variedad de aplicaciones.
3. XML debe ser compatible con SGML.
4. Ha de resultar fácil escribir programas que procesen documentos XML.

ANOTACIONES

5. El número de características opcionales en XML debe ser mantenido en un mínimo, idealmente cero.
6. Los documentos XML deben ser legibles por un ser humano y razonablemente claros.
7. El diseño de XML debe ser preparado rápidamente.
8. El diseño de XML debe ser formal y conciso.
9. Los documentos XML han de ser fáciles de crear.
10. La brevedad en la marcación reviste mínima importancia.

En otro punto del estándar se referencian además los otros estándares relacionados. Éstos, junto con la definición de XML, son cuanto se necesita para comprender XML.



Esta especificación, junto con los estándares asociados (Unicode and ISO/IEC 10646 para caracteres, Internet RFC 1766 para las marcas de identificación de lenguaje, ISO 639 para los códigos de nombre de lenguaje, ISO 3166 para los códigos de nombre de país), proporciona toda la información necesaria para entender XML Versión 1.0 y construir programas de computador que lo procesen.

Cabe preguntarse qué llevó al W3C a desarrollar un nuevo lenguaje para la web cuando ya disponíamos de HTML. En 1996 se habían puesto de manifiesto algunas de las más destacadas carencias de HTML:

- HTML estaba optimizado para ser fácil de aprender, no para ser fácil de procesar:
  - Un solo conjunto de marcas (independientemente de las aplicaciones).

ANOTACIONES

– Una semántica predefinida para cada marca.

– Estructuras de datos predefinidas.

• HTML sacrifica la potencia para conseguir facilidad de uso.

• HTML resulta adecuado para aplicaciones simples, pero es poco adecuado para aplicaciones complejas:

– Conjuntos de datos complejos.
– Datos que deben ser manipulados de formas diversas.
– Datos para controlar programas.
– Sin capacidad de validación formal.

Ante todo esto, el W3C desarrolló un nuevo lenguaje (XML), que nos proporciona:

Extensibilidad: se pueden definir nuevas marcas y atributos según sea necesario.

Estructura: se puede modelizar cualquier tipo de datos que esté organizado jerárquicamente.

Validez: podemos validar automáticamente los datos (de forma estructural).

Independencia del medio: podemos publicar el mismo contenido en multitud de medios.

Podemos definir XML de la siguiente manera:



Definición

XML es una versión simplificada de SGML muy fácil de implementar, que no constituye un lenguaje, sino un meta-lenguaje diseñado para permitir la definición de

ANOTACIONES

un número ilimitado de lenguajes para propósitos específicos, pero que pueden ser procesados empleando las mismas herramientas independientemente del propósito para el que se han construido.

Como podemos ver en la definición, XML no es un lenguaje, sino un meta-lenguaje que nos permite definir multitud de lenguajes para propósitos específicos. En los capítulos siguientes iremos viendo cómo definir estos lenguajes y cómo definir las normas de validación estructural de los lenguajes. Estas definiciones, e incluso las definiciones de *programas* de traducción y transformación de ficheros XML, se definen en XML, lo que da una muestra de la potencia de XML. En XML se ha definido gran número de lenguajes. De hecho, el fichero de configuración de algunos de los programas más usados en la web (Tomcat, Roxen, etc.) está definidos con XML. Hay muchos ficheros de datos, de documentos, etc., que también lo usan para su definición. Algunos de los lenguajes definidos con XML más conocidos son:

- **SVG** (*scalable vector graphics*).
- **DocBook XML** (Docbook-XML).
- **XMI** (XML *metadata interface format*).
- **WML** (WAP *markup language*).
- **MathML** (*mathematical markup language*).
- **XHTML** (XML *hypertext markup language*).

Podemos ver un ejemplo de documento XML definido por nosotros:

```
<?xml version="1.0" encoding="iso-8859-1"?>
 <biblioteca>
   <libro idioma="inglés">
      <titulo>The Hobbit</titulo>
      <autor>J. R. R. Tolkien</autor>
      <editorial>Allen and Unwin</editorial>
   </libro>
   <libro idioma="catellano">
      <titulo>El Quijote</titulo>
```

ANOTACIONES

```
      <autor>Miguel de Cervantes</autor>
      <editorial>Alfaguara</editorial>
   </libro>
</biblioteca>
```

Podemos apreciar en este documento del ejemplo algunas de las características ya mencionadas de XML, como su estructura jerárquica, su legibilidad o el hecho de que está diseñado para un uso concreto (almacenar los libros de nuestra biblioteca). Este ejemplo también nos muestra las diferencias entre XML y HTML que justifican el nacimiento de XML. Escribamos en HTML un documento parecido al anterior:

```
<HTML>
 <HEAD>
    <TITLE>Biblioteca</TITLE>
 </HEAD>
 <BODY>
    <H1>The Hobbit</H1>
    <P><B>Autor:</B>J. R. R. Tolkien</P>
    <P><B>Editorial</B>Allen and Unwin</P>

    <H1>El Quijote</H1>
    <P><B>Autor:</B>Miguel de Cervantes</P>
    <P><B>Editorial</B>Alfaguara</P>
 </BODY>
</HTML>
```

Como podemos apreciar, en HTML tenemos medios para representar el formato de visualización (<B>, <I>, etc., representan formato: negrillas, itálicas, etc.), pero éstos no proporcionan información sobre la semántica de los datos contenidos. En XML disponemos de etiquetas que nos informan de la semántica de los datos: <autor> nos indica que el dato contenido corresponde al autor, etc. No obstante, no nos proporcionan ninguna información sobre el formato de visualización, ni nos dice cómo pintar el nombre del autor en pantalla (si en negrillas, en itálicas, etc.). Sin embargo, XML proporciona otras herramientas que nos permiten especificar esos puntos permitiéndonos cambiar el formato de representación en función de dónde lo vayamos a mostrar (adaptándolo a un navegador web concreto, a un teléfono móvil, etc.).

ANOTACIONES

4.2. XML

Un objeto XML (o un documento XML) se define como un documento formado por etiquetas y valores que cumple con la especificación de XML, y que estará bien formado.

Iniciaremos el estudio de XML observando un ejemplo de documento XML para estudiar las partes que lo componen. A tal fin diseñaremos un formato XML para representar recetas de cocina. Con este formato XML (generalmente denominado aplicación XML), al que llamaremos RecetaXML, guardaremos nuestras recetas.

```
<?xml version="1.0"?>
  <Receta>
   <Nombre>Tortilla de patatas</Nombre>
   <Descripcion>
    La tradicional y típica tortilla de patatas,
    tal como la hacen todas las madres.
   </Descripcion>
   <Ingredientes>
      <Ingrediente>
         <Cantidad unidad="pieza">3</Cantidad>
         <Item>Patata</Item>
      </Ingrediente>
      <Ingrediente>
         <Cantidad unidad="pieza">2</Cantidad>
         <Item>Huevos</Item>
      </Ingrediente>
      <Ingrediente>
         <Cantidad unidad="litro">0.1</Cantidad>
         <Item>Aceite</Item>
      </Ingrediente>
      </Ingredientes>
      <Instrucciones>
         <Paso>
Pelar y cortar la patata en rodajas
         </Paso>
         <Paso>
Poner aceite en una paella
         </Paso>
         <!-- Y así seguimos... -->
      </Instrucciones>
</Receta>
```

Nota

Podemos encontrar la especificación XML en el sitio web del W3C: http://www.w3c.org

ANOTACIONES

Receta escrita en RecetaXML

Este documento es un documento XML. Todos los documentos XML *bien formados* deben iniciarse con una línea como:

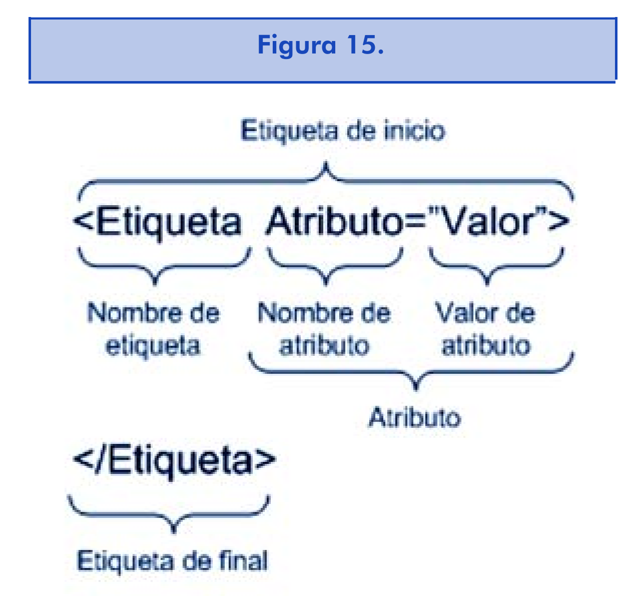
```
<?xml version="1.0"?>
```

Dicha línea nos indica la versión de la especificación XML utilizada, informándonos de que se trata de un documento XML. Además, debe componerse exclusivamente de etiquetas XML organizadas jerárquicamente. Podemos ver fácilmente que XML no almacena cómo debe representarse o mostrarse la información, sino que almacena la semántica de ésta. De este documento podemos deducir cómo se organiza la información de la misma:



Una receta está formada por una lista de ingredientes y unas instrucciones. La lista de ingredientes es un conjunto de ingredientes donde tenemos, por cada ingrediente, un nombre, una cantidad, etc.

Figura 15.



Como podemos apreciar en nuestra receta, todas las etiquetas de XML siguen el formato mostrado.

ANOTACIONES

ANOTACIONES

4.2.1. Documento bien formado

El concepto de *bien formado* procede de las matemáticas, donde es factible escribir una expresión usando símbolos matemáticos como:

1)1(- - 5(+ =)4 < 3

que, a pesar de estar formada por símbolos matemáticos, no significa nada, ya que no sigue las convenciones y normas de escritura de expresiones matemáticas. Esta expresión matemática no está *bien formada*.

En XML, un documento bien formado es el que sigue las siguientes normas:

Todas las etiquetas cerradas: en HTML podemos *trabajar* con un gran nivel de relajación de las normas sintácticas, lo cual nos permite dejar etiquetas (como <B> por ejemplo) abiertas a lo largo de todo el documento, o usando indiscriminadamente etiquetas como <P> sin cerrarlas con la correspondiente </P>. XML no permite este nivel de relajación. Todas las etiquetas abiertas deben tener su correspondiente etiqueta de cierre. Esto se debe a que las etiquetas en XML representan una información jerárquica que nos indica cómo se relacionan los diferentes elementos entre ellos. Si no cerramos las etiquetas, introducimos en esta representación una serie de ambigüedades que nos impedirían un procesamiento automático.

No puede haber solapamiento de etiquetas: una etiqueta que se abre dentro de otra etiqueta debe cerrarse antes de cerrar la etiqueta contenedora. Por ejemplo:

```
<Libro>Platero y Yo<Autor>J. R.Jiménez</Libro></Autor>
```

No está bien formado porque **Autor** no se cierra dentro de Libro, que es donde debería cerrarse. La sentencia correcta sería:

```
<Libro>Platero y Yo<Autor>J. R. Jiménez</Autor></Libro>
```

Es decir, la estructura del documento debe ser estrictamente jerárquica.

Los valores de los atributos deben estar entrecomillados; a diferencia de HTML, donde podemos poner atributos sin comillas. Por ejemplo,

```
<IMAGE SRC=img.jpg SIZE=10>
```

en XML todos los atributos deben estar entrecomillados. Los atributos anteriores quedarían, pues, de la siguiente manera:

```
<IMAGE SRC="img.jpg" SIZE="10">.
```

Los caracteres <, > y " siempre representados por entidades de carácter: para representar estos caracteres (en el texto, no como marcas de etiqueta) debemos usar siempre las entidades de carácter especiales: **<**, **>** y **"**. Estos caracteres son especiales para XML.

4.2.2. Bien formado equivale a analizable

La importancia que reviste el hecho de que un documento esté o no bien formado en XML deriva del hecho de que un documento bien formado puede ser analizable sintácticamente o *parseable* (es decir, procesable automáticamente). Existen multitud de *parsers* (analizadores) en numerosos lenguajes de programación que nos permiten trabajar con los documentos XML. Los *parsers* de XML son capaces de detectar los errores estructurales de los documentos XML (es decir, si están o no bien formados) y notificárselo al programa. Esta funcionalidad es importantísima para un programador, ya que le libera de la tarea de detectar errores para encomendársela a un programa que lo hará por sí solo (el *parser*).

Algunos *parsers* van más allá de la simple capacidad de detectar si el documento está bien formado o no y son capaces de identificar los documentos *válidos*, entendiendo por válido el hecho de que la estructura, posición y número de etiquetas sean correctos y tengan sentido. Imaginemos por un momento el siguiente pedazo de nuestro documento de recetas:

```
<Ingrediente>
   <Cantidad unidad="pieza">3</Cantidad>
   <Cantidad unidad="litro">4</Cantidad>
   <Item>Patatas</Item>
</Ingrediente>
```

Este XML está bien formado, sigue todas las normas para que lo esté, pero a pesar de ello no tiene ningún sentido. ¿Que significaría? Tenemos **patatas** en la receta, ¿pero en qué cantidad? En este caso el problema sería que tenemos un documento XML bien formado, pero que carece de utilidad, ya que no tiene sentido. Necesitamos indicar de alguna manera cómo controlar que un documento tenga sentido. En nuestro caso, tenemos que especificar que cada ingrediente tendrá sólo una etiqueta de tipo cantidad, que ésta tendrá un atributo opcional unidad y que no contendrá etiquetas anidadas. Para ello, XML nos proporciona un par de lenguajes de especificación de estructura del documento, **XML Schema** y **DTD**, que veremos posteriormente.

4.2.3. Espacios de nombres

XML es un estándar diseñado para permitir que se comparta información con facilidad. ¿Qué pasaría si uniésemos información en XML procedente de dos fuentes diferentes para enviarla a una tercera? ¿Podríamos en ese caso tener algún conflicto por coincidencia del nombre de las etiquetas?

Imaginemos el siguiente caso: un proveedor de Internet guarda todos sus datos en XML. La sección comercial almacena la dirección de la vivienda del cliente en un campo llamado `<direccion>`. Por otro lado, el servicio de asistencia al cliente guarda en `<direccion>` la dirección de correo electrónico del cliente, y finalmente el centro de control de red guarda en `<direccion>` la dirección IP del ordenador del cliente. Si unimos en un sólo fichero lo procedente de las tres divisiones de la empresa, nos podemos encontrar con:

```
<cliente>
    ...
    <direccion>Calle Real</direccion>
    ...
    <direccion>ventas@cliente.com</direccion>
    ...
    <direccion>192.168.168.192</direccion>
    ...
</cliente>
```

Evidentemente en este caso tendríamos un problema, ya que no podríamos distinguir el significado de `<direccion>` en cada uno de los casos.

ANOTACIONES

Para ello, en 1999 el W3C definió una extensión de XML llamada espacios de nombres (*namespaces*) que permite resolver conflictos y ambigüedades de este tipo.

Uso de los espacios de nombres

Los espacios de nombres son un prefijo que ponemos a las etiquetas de XML para indicar a qué contexto se refiere la etiqueta en cuestión. En el ejemplo anterior podríamos definir:

<**red:direccion**>: para uso por el centro de control de red.
<**aten:direccion**>: para uso por el servicio de atención al cliente.
<**comer:direccion**>: para uso por el departamento comercial.

De esta forma, nuestro elemento queda compuesto así:

```
<cliente>
 ...
 <comer:direccion>Calle Real</comer:direccion>
 ...
 <aten:direccion>ventas@cliente.com</aten:direccion>
 ...
 <red:direccion>192.168.168.192</red:direccion>
 ...
</cliente>
```

Para usar un espacio de nombres en un documento, debemos declararlo previamente. Esta declaración puede tener lugar en el elemento raíz del documento de la siguiente forma:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<carteraclientes
    xmlns:comer="http://www.empresa.com/comercial"
    xmlns:aten="http://www.empresa.com/atencion"
    xmlns:red="http://www.empresa.com/red">
 <cliente>
    ...
    <comer:direccion>Calle Real</comer:direccion>
    ...
    <aten:direccion>ventas@cliente.com</aten:direccion>
    ...
    <red:direccion>192.168.168.192</red:direccion>
    ...
 </cliente>
</carteraclientes>
```

ANOTACIONES

La definición consta de unos atributos `xmlns` (*XML namespace*), donde proporcionamos el prefijo que usaremos para el espacio de nombres y una URI (*Uniform Resource Identifier*) que será un identificador único del espacio de nombres.

4.3. Validación: DTD y XML Schema

Como hemos visto, XML posibilita la comprobación automática de la correcta forma de un documento, pero sin información adicional es imposible comprobar la validez de éste a partir del propio documento. Para ello, el W3C ha desarrollado algunos estándares de XML que nos permiten validar un documento a partir de una especificación formal de cómo debe ser éste. Dichos estándares son DTD y XSchema.

DTD es un estándar antiguo, derivado de SGML y que adolece de algunas deficiencias graves, siendo la más grave de ellas el hecho de no estar escrito en XML. XSchema, por otro lado, es un estándar relativamente moderno, muy potente y extensible, que además está escrito en XML íntegramente.

4.3.1. DTD

DTD (***Document Type Definition***) es un estándar que nos permite definir una *gramática* que deben cumplir nuestros documentos XML para considerarlos válidos. Una definición DTD para *n* documentos XML especifica: qué elementos pueden existir en un documento XML, qué atributos pueden tener éstos, qué elementos pueden o deben aparecer contenidos en otros elementos y en qué orden.

Los *parsers* de XML que son capaces de validar documentos con DTD leen esos documentos y el DTD asociado. En caso de que el documento XML no cumpla los requerimientos que le impone el DTD, nos advertirán del error y no validarán el documento.

Mediante los DTD definimos cómo será nuestro dialecto de XML (recordad que nosotros definimos qué etiquetas vamos a usar en nuestros documentos, qué significado les damos, etc.). Esta capacidad de definir un dialecto propio de XML es lo que permite que XML se denomine. A pesar

ANOTACIONES

de que DTD es un estándar que deberá ser sustituido por XML Schema, sigue siendo muy usado. Además, su uso resulta más simple que el de XML Schema. Por otro lado, es más compacto. A eso hay que añadir que las mejoras que aporta XML Schema no son necesarias para la mayoría de los usos. Con DTD se han definido multitud de dialectos de XML que son usados ampliamente en Internet, como RDF para la web semántica, MathML para documentos matemáticos, XML/EDI para intercambio de datos electrónicamente para negocio, VoiceXML para aplicaciones que se utilicen mediante voz o que hagan uso de ésta, WML para representar documentos para los navegadores de dispositivos móviles como teléfonos, etc.

Veamos un posible DTD para la receta del ejemplo que nos definirá la forma que deben tener las recetas escritas en RecetaXML:

```
<!-- DTD de ejemplo para RecetaXML -->
<!ELEMENT Receta (Nombre, Descripcion?,
    Ingredientes?, Instrucciones?)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Descripcion (#PCDATA)>
<!ELEMENT Ingredientes (Ingrediente*)>
<!ELEMENT Ingrediente (Cantidad, Item)>
<!ELEMENT Cantidad (#PCDATA)>
<!ATTLIST Cantidad unidad CDATA #REQUIRED>
<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item opcional CDATA "0"
    vegetariano CDATA "si">
<!ELEMENT Instructiones (Paso+)>
<!ELEMENT Paso (#PCDATA)>
```

De este documento DTD podemos inferir una descripción de las reglas de validez que sea un poco más legible:

- Una receta consta de un nombre (obligatorio), una descripción (opcional), unos ingredientes (opcionales) y unas instrucciones (opcionales).
- El nombre y la descripción pueden contener caracteres alfanuméricos (PCDATA corresponde a *Parsed Character Data*).
- Los ingredientes son una lista de elementos *ingrediente*.
- Un ingrediente consta de un ítem y la cantidad.

ANOTACIONES

- La cantidad es un valor alfanumérico, teniendo la etiqueta un atributo, unidad que nos describe qué unidad de medida estamos utilizando.
- Un ítem de la receta consta del nombre (un valor alfanumérico) y puede tener dos atributos: opcional (si el ingrediente es o no obligatorio) y vegetariano (si el ingrediente es apto para vegetarianos).
- Las instrucciones de elaboración son una lista de pasos.
- Un paso consta de un texto alfanumérico descriptivo del paso.

Vamos a estudiar ahora la sintaxis de DTD para definir los dialectos XML.

Convenciones sintácticas de DTD

Como hemos visto, la sintaxis de DTD no resulta evidente a primera vista. Pese a ello, tampoco es excesivamente compleja. El primer paso para entenderla es disponer de las definiciones y usos de los diferentes símbolos usados, que podemos ver en la tabla siguiente:

Tabla 5. Elementos sintácticos de DTD

Símbolo	Descripción
()	Los paréntesis agrupan subetiquetas <!ELEMENT Ingrediente (Cantidad,Item)>
,	Ordenación exacta de los elementos (Nombre, Descripcion?, Ingredientes?, Instrucciones?)
|	Uno sólo de los elementos indicados (Cocer | Freir)
	Si no indicamos nada los elementos aparecen una sola vez (Cantidad, Item)
+	Una o más veces Paso+
?	Elemento opcional Instrucciones?
*	Cero o más veces Ingrediente*
#PCDATA	Parsed Character Data <!ELEMENT Item (#PCDATA)>

Elemento ELEMENT

Los elementos de DTD llamados ELEMENT nos definen una etiqueta de nuestro dialecto de XML. Por ejemplo:

```
<!ELEMENT Receta (Nombre, Descripcion?,
     Ingredientes?, Instrucciones?)>
```

Define la etiqueta Receta, especificando qué contienen las subetiquetas: Nombre, Descripción, Ingredientes e Instrucciones, y agregando que estas tres últimas son opcionales (como indica el símbolo ?).

La definición de ELEMENT es la siguiente:

```
<!ELEMENT nombre categoria>
<!ELEMENT nombre (contenido)>
```

- **Elementos vacíos**

  Los elementos vacíos se declaran empleando la categoría EMPTY.

  ```
  <!ELEMENT nombre EMPTY>
  ```

  Este elemento nombre, en XML se usaría así:

  ```
  <nombre />
  ```

- **Elementos con sólo caracteres**

  Los elementos que sólo contendrán datos alfanuméricos se declaran usando #PCDATA entre paréntesis.

  ```
  <!ELEMENT nombre (#PCDATA)>
  ```

- **Elementos con cualquier contenido**

  Los elementos que declaremos usando ANY como indicador de contenido pueden contener cualquier combinación de datos *parseables*:

  ```
  <!ELEMENT nombre ANY>
  ```

ANOTACIONES

• **Elementos con subelementos (secuencias)**

Los elementos con uno o más elementos hijos se definen con el nombre de los elementos hijos entre paréntesis:

```
<!ELEMENT nombre (hijo1)>
<!ELEMENT nombre (hijo1, hijo2, ......)>
```

Por ejemplo:

```
<!ELEMENT coche (marca, matricula, color)>
```

Los hijos que se declaran como una secuencia de elementos separados por comas deben aparecer en el mismo orden en el documento. Los elementos hijo también deben declararse en el documento DTD. Estos elementos hijo pueden, a su vez, tener elementos hijo. La declaración completa de coche sería entonces:

```
<!ELEMENT coche (marca, matricula, color)>
<!ELEMENT marca (#PCDATA)>
<!ELEMENT matricula (#PCDATA))>
<!ELEMENT color (#PCDATA)>
```

• **Cardinalidad de las ocurrencias de elementos**

La siguiente declaración nos indica que el elemento hijo sólo puede ocurrir una vez dentro del elemento padre:

```
<!ELEMENT nombre (hijo)>
```

Si deseamos que el elemento hijo aparezca más de una vez y como mínimo una vez:

```
<!ELEMENT nombre (hijo+)>
```

Si deseamos que pueda aparecer cualquier número de veces (incluyendo la posibilidad de que no aparezca ninguna):

```
<!ELEMENT nombre (hijo*)>
```

Si sólo deseamos que pueda aparecer una vez, pero que no sea obligatorio:

```
<!ELEMENT nombre (hijo?)>
```

ANOTACIONES

- **Elementos con contenido mixto.**

  Podemos declarar también elementos que contengan otros elementos hijos y/o datos alfanuméricos.

```
<!ELEMENT nombre (#PCDATA hijo hijo2)*>
```

Elemento ATTLIST

Como ya hemos visto, los elementos en XML pueden tener atributos. Evidentemente, en DTD disponemos de un mecanismo para indicar qué atributos puede tener un ELEMENT, de qué tipo, si son o no obligatorios, etc. Para ello disponemos del elemento ATTLIST, cuya sintaxis es:

```
<!ATTLIST elemento atributo tipo-atributo valor-defecto>
```

Un ejemplo de uso sería:

```
<!ATTLIST pago metodo CDATA "contra-reembolso" >
```

Y su uso en XML:

```
<pago metodo="contra-reembolso" />
```

El tipo de atributo debe ser uno de los de la lista:

Tabla 6. Atributos

Valor	Descripción
CDATA	El valor son caracteres alfanuméricos
(v1 | v2 | ..)	El valor será uno de la lista explicitada
ID	El valor será un identificador único
IDREF	El valor es el ID de otro elemento
IDREFS	El valor es una lista de ID otros elementos
NMTOKEN	El valor es un nombre XML válido
NMTOKENS	El valor es una lista de nombres XML válidos
ENTITY	El valor es una entidad
ENTITIES	El valor es una lista de entidades
NOTATION	El valor es el nombre de una notación
xml:	El valor es un valor XML predefinido

ANOTACIONES

ANOTACIONES

El valor por defecto puede ser uno de los siguientes:

Tabla 7. Valores

Valor	Descripción
valor	El valor por defecto del atributo
#REQUIRED	El valor del atributo debe aparecer obligatoriamente en el elemento
#IMPLIED	El atributo no tiene por qué ser incluido
#FIXED valor	El valor del atributo es fijo

- **Valor por defecto**

  En el siguiente ejemplo:

```
<!ELEMENT pago EMPTY>
<!ATTLIST pago metodo CDATA "contra-reembolso" >
```

  El siguiente XML se considera válido:

```
<pago />
```

  En este caso, donde no especificamos valor para método, éste contendrá el valor por defecto de contra-reembolso.

- **Sintaxis de #IMPLIED**

  En el siguiente ejemplo:

```
<!ELEMENT pago EMPTY>
<!ATTLIST pago metodo CDATA #IMPLIED >
```

  Validará correctamente el siguiente XML:

```
<pago metodo="tarjeta" />
<pago />
```

  Usaremos, pues, #IMPLIED en aquellos casos en los que no queremos forzar al usuario a usar atributos pero no podemos poner valores por defecto.

• Sintaxis de #REQUIRED

En el siguiente ejemplo:

```
<!ELEMENT pago EMPTY>
<!ATTLIST pago metodo CDATA #REQUIRED >
```

Validará correctamente el siguiente XML:

```
<pago metodo="tarjeta" />
```

pero no validará:

```
<pago />
```

Usaremos #REQUIRED en aquellos casos en los que no podemos proporcionar un valor por defecto, pero deseamos que el atributo aparezca y se le asigne algún valor.

Vincular un DTD con un documento

Para que un documento XML quede vinculado a un DTD determinado, tenemos dos opciones: incluir el DTD en el documento XML o usar una referencia externa al DTD.

La primera opción es la más fácil de usar, pero la que presenta más inconvenientes, ya que aumenta el tamaño de los documentos XML y complica su mantenimiento, puesto que un cambio en el DTD implica revisar todos los documentos en los que lo hemos incluido.

El formato de un documento XML donde hubiésemos incluido sería:

```
<?xml version="1.0"?>
<!DOCTYPE Receta [
    <!ELEMENT Receta (Nombre, Descripcion?,
       Ingredientes?, Instrucciones?)>
    <!ELEMENT Nombre (#PCDATA)>
    <!ELEMENT Descripcion (#PCDATA)>
    <!ELEMENT Ingredientes (Ingrediente)*>
    <!ELEMENT Ingrediente (Cantidad, Item)>
    <!ELEMENT Cantidad (#PCDATA)>
```

ANOTACIONES

```
    <!ATTLIST Cantidad unidad CDATA #REQUIRED>
    <!ELEMENT Item (#PCDATA)>
    <!ATTLIST Item opcional CDATA "0"
       vegetariano CDATA "si">
    <!ELEMENT Instructiones (Paso)+>
    <!ELEMENT Paso (#PCDATA)>
]>
<Receta>
   <Nombre>Tortilla de patatas</NOMBRE>
   <Descripcion>
     La tradicional y típica tortilla de patatas,
     tal como la hacen todas las madres.
   </Descripcion>
   <Ingredientes>
     <Ingrediente>
       <Cantidad unidad="pieza">3</Cantidad>
       <Item>Patata</Item>
    </Ingrediente>
    <Ingrediente>
      <Cantidad unidad="pieza">2</Cantidad>
      <Item>Huevos</Item>
    </Ingrediente>
    <Ingrediente>
       <Cantidad unidad="litro">0.1</Cantidad>
      <Item>Aceite</Item>
    </Ingrediente>
   </Ingredientes>
   <Instrucciones>
       <Paso>
Pelar y cortar la patata en rodajas
       </Paso>
       <Paso>
Poner aceite en una paella
       </Paso>
       <!-- ... ... -->
   </Instrucciones>
</Receta>
```

Podemos referenciar un DTD externo al documento XML. Para ello disponemos de dos tipos de referencias posibles: públicas o privadas.

Un ejemplo de referencia privada es el siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE Receta SYSTEM "receta.dtd">
<Receta>
...
```

ANOTACIONES

Y otro, usando ahora una referencia externa pública:

```
<?xml version="1.0"?>
<!DOCTYPE Receta
   PUBLIC "-//W3C//DTD XHTML 1.0 STRICT/EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<Receta>
...
```

4.3.2. XML Schema

XML Schema nace en el 2001, procedente de los esfuerzos del W3C para paliar las deficiencias evidentes de DTD:

- No tenemos un gran nivel de control sobre qué se considera válido.
- No disponemos de control de tipos de datos (enteros, etc.).
- No está definido como XML.
- La sintaxis en algunas ocasiones es un poco compleja.

XML Schema presenta ciertas características que la hacen mucho más potente que DTD:

- Está definida en XML, lo cual permite validar también los documentos XML Schema.
- Permite control sobre los tipos de datos (enteros, etc.).
- Permite definir nuevos tipos de datos.
- Permite describir el contenido de los documentos.
- Facilita validar que los datos son correctos.
- Facilita definir patrones (formatos) de datos.

A pesar de sus numerosas ventajas, DTD continua siendo el mecanismo más usado para definir la estructura de los documentos XML.

ANOTACIONES

- Extensibilidad

XML Schema es ampliable, ya que permite reutilizar desde un esquema las definiciones procedentes de otro esquema y definir tipos de datos a partir de los tipos de datos procedentes del estándar, así como de otros esquemas y además hace posible que un documento use varios esquemas.

Introduciremos XML Schema (también llamado XSD, XML *schema definition*) comparándolo con el conocido DTD. Partiremos de un documento XML, nuestro documento de RecetaXML.

```
<?xml version="1.0"?>
  <Receta>
    <Nombre>Tortilla de patatas</Nombre>
    <Descripcion>
     La tradicional y típica tortilla de patatas,
     tal como la hacen todas las madres.
    </Descripcion>
    <Ingredientes>
      <Ingrediente>
         <Cantidad unidad="pieza">3</Cantidad>
         <Item>Patatas</Item>
      </Ingrediente>
      <Ingrediente>
         <Cantidad unidad="pieza">2</Cantidad>
         <Item>Huevos</Item>
      </Ingrediente>
      <Ingrediente>
         <Cantidad unidad="litro">0.1</Cantidad>
         <Item>Aceite</Item>
     </Ingrediente>
    </Ingredientes>
    <Instrucciones>
         <Paso>
Pelar y cortar la patata en rodajas
         </Paso>
         <Paso>
Poner aceite en una paella
         </Paso>
         <!-- Y así seguimos... -->
    </Instrucciones>
</Receta>
```

ANOTACIONES

Esta vez, en lugar de mostrar el DTD asociado, que ya vimos en la sección anterior, definiremos un XSD para el documento en RecetaXML.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
   <xs:element name="Cantidad">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:decimal">
          <xs:attribute name="unidad"
            type="xs:string" use="required"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="Descripcion" type="xs:string"/>
      <xs:element name="Ingrediente">
        <xs:complexType>
         <xs:sequence>
           <xs:element ref="Cantidad"/>
           <xs:element ref="Item"/>
         </xs:sequence>
        </xs:complexType>
 </xs:element>
        <xs:element name="Ingredientes">
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="Ingrediente"
            maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="Instrucciones">
        <xs:complexType>
         <xs:sequence>
           <xs:element ref="Paso"
              maxOccurs="unbounded"
              minOccurs="1"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="Item" type="xs:string" />
       <xs:element name="Nombre" type="xs:string"/>
       <xs:element name="Paso" type="xs:string"/>
       <xs:element name="Receta">
        <xs:complexType>
         <xs:sequence>
           <xs:element ref="Nombre"/>
           <xs:element ref="Descripcion"/>
           <xs:element ref="Ingredientes"/>
           <xs:element ref="Instrucciones"/>
         </xs:sequence>
        </xs:complexType>
      </xs:element>
</xs:schema>
```

ANOTACIONES

El elemento <schema>

El elemento <schema> es el elemento raíz de todos los XSD:

```
<?xml version="1.0"?>
<xs:schema>
... ...
</xs:schema>
```

El elemento puede tener algunos atributos; de hecho, generalmente aparece de una forma similar a ésta:

```
   <?xml version="1.0"?> <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.empresa.com"
      xmlns="http://www.empresa.com"
          elementFormDefault="qualified">
      ... ...
</xs:schema>
```

El siguiente fragmento:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

indica que los elementos y tipos de datos usados en nuestro esquema (schema, element, etc.) provienen del espacio de nombres definido en http://www.w3.org/-2001/ XMLSchema. Además, asigna a este espacio de nombres un prefijo, en este caso, xs.

El fragmento targetNamespace="http://www.empresa.com" indica que los elementos definidos por este esquema (Ingrediente, etc.) provienen del espacio de nombres de "http://www.empresa.com".

Este otro, xmlns="http://www.empresa.com", indica que el espacio de nombres por defecto es el de http://www.empresa.com, permitiéndonos así asignar las etiquetas que no tienen prefijo a este espacio de nombres.

Este otro, elementFormDefault="qualified" indica que todos los elementos usados en el documento XML que se declaren en este esquema deberán clasificarse dentro del espacio de nombres.

Elementos simples

Un elemento simple es aquel elemento XML que sólo puede contener texto, sin posibilidad de contener otros elementos ni atributos. A pesar de ello, el texto contenido puede ser de cualquiera de los tipos incluidos en la definición de XSD (booleano, entero, cadena, etc.) o puede ser un tipo definido por nosotros.

Podemos, además, añadir restricciones para limitar los contenidos o bien podemos requerir que los datos contenidos sigan un patrón determinado.

Para definir un elemento simple, la sintaxis que emplearemos será:

```
<xs:element name="nombre" type="tipo"/>
```

Donde *nombre* es el nombre del elemento y *tipo* es el tipo de datos del elemento. Algunos ejemplos de declaraciones:

```
<xs:element name="Item" type="xs:string" />
<xs:element name="Edad" type="xs:integer"/>
<xs:element name="Fecha" type="xs:date"/>
```

Aquí tenemos algunos elementos de XML que cumplen las restricciones anteriores:

```
<Item>Patatas</Item>
<Edad>34</Edad>
<Fecha>1714-09-11</Fecha>
```

XSD nos proporciona los siguientes tipos de datos:

– xs:string
– xs:decimal
– xs:integer
– xs:boolean
– xs:date
– xs:time

ANOTACIONES

- **Valores fijos y valores por defecto**

  Los elementos simples pueden tener un valor por defecto o un valor prefijado. Un valor por defecto se asigna automáticamente al elemento cuando no especificamos un valor. El ejemplo asigna Aceites como valor por defecto.

```
<xs:element name="Item" type="xs:string"
  default="Aceites"/>
```

  Un elemento con valor prefijado siempre tiene el mismo valor y no podemos asignarle ningún otro.

```
<xs:element name="Item" type="xs:string"
  fixed="Aceites"/>
```

Atributos

Los elementos simples no pueden tener atributos. Si un elemento tiene atributos, se considera que es de tipo complejo. El atributo, por otro lado, se declara siempre como tipo simple. Disponemos para los atributos de los mismos tipos de datos básicos que para los elementos.

La sintaxis para definir un atributo es:

```
<xs:attribute name="nombre" type="tipo"/>.
```

Donde *nombre* es el nombre del atributo y *tipo*, el tipo de datos de este. Un elemento XML con atributos de RecetaXML es el siguiente:

```
<Cantidad unidad="pieza">3</Cantidad>
```

Y el XSD correspondiente a su definición:

```
<xs:attribute name="unidad" type="xs:string"
   use="required"/>
```

ANOTACIONES

• Valores fijos y valores por defecto

La declaración de atributos con valores prefijados y valores por defecto sigue el mismo esquema que para los elementos:

```
<xs:attribute name="unidad" type="xs:string"
   default="gramos"/>
<xs:attribute name="unidad" type="xs:string"
   fixed="gramos"/>
```

• Atributos opcionales y obligatorios

Los atributos son opcionales por defecto. Podemos, no obstante, especificar explícitamente que el atributo es opcional:

```
<xs:attribute name="unidad" type="xs:string"
   use="optional"/>
```

Para explicitar que éste es obligatorio:

```
<xs:attribute name="unidad" type="xs:string"
   use="required"/>
```

Restricciones al contenido

Con XSD podemos ampliar las restricciones al contenido que nos proporcionan los tipos de datos de XSD (enteros, etc.) con otras restricciones diseñadas por nosotros (*facets*). Por ejemplo, en el siguiente código especificamos un elemento, la edad, que debe tener un valor entero entre 1 y 120.

```
<xs:element name="edad">
<xs:simpleType>
 <xs:restriction base="xs:integer">
   <xs:minInclusive value="1"/>
   <xs:maxInclusive value="120"/>
 </xs:restriction>
</xs:simpleType>
</xs:element>
```

ANOTACIONES

ANOTACIONES

- **Restricciones a un conjunto de valores.**

  Podemos limitar con XSD el contenido de un elemento de XML de tal manera que sólo pueda contener un valor perteneciente a un conjunto de elementos aceptables, usando para ello la restricción de enumeración *(enumeration)*. Por ejemplo, definimos un elemento llamado *vino* especificando los valores posibles:

```
<xs:element name="vino">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:enumeration value="Blanco"/>
     <xs:enumeration value="Rosado"/>
     <xs:enumeration value="Tinto"/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

  El elemento *vino* anterior es un tipo de datos simple con restricciones, siendo sus valores aceptables: *Tinto*, *Rosado* y *Blanco*. También podríamos haberlo definido de la siguiente forma:

```
<xs:element name="vino" type="tipoVino"/>
<xs:simpleType name="tipoVino">
   <xs:restriction base="xs:string">
     <xs:enumeration value="Blanco"/>
     <xs:enumeration value="Rosado"/>
     <xs:enumeration value="Tinto"/>
   </xs:restriction>
</xs:simpleType>
```

  Cabe destacar que, en este caso, podríamos usar el tipo de datos `tipoVino` para otros elementos, ya que no es parte de la definición de `vino`.

- **Restricciones a una serie de valores**

  Podemos usar patrones para definir elementos que contengan una serie de números o letras concreta. Por ejemplo:

```
<xs:element name="letra">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="[a-z]"/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

El elemento **letra** es un tipo simple con restricción cuyo único valor aceptable es **UNA** de las letras minúsculas.

Por ejemplo, el siguiente elemento:

```
<xs:element name="codigoproducto">

<xs:simpleType>
   <xs:restriction base="xs:integer">
     <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

Define un **codigoproducto** formado por exactamente cinco dígitos, del 0 al 9.

El siguiente código define un tipo **letras**:

```
<xs:element name="letras">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="([a-z])*"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

que puede tomar por valor cualquier letra minúscula que aparezca cero o más veces, es decir, puede tener un valor nulo. Sin embargo, el siguiente:

```
<xs:element name="letras">

<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="([a-z][A-Z])+"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

permite que el elemento contenga letras mayúsculas, pero obliga a que contenga una letra como mínimo.

Los patrones nos permiten el mismo tipo de definiciones que las restricciones de conjunto de elementos, por ejemplo:

ANOTACIONES

```
<xs:element name="sexo">

<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="hombre mujer"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

Nos define un elemento **sexo** que puede tomar como valor: *hombre* o *mujer*. Podemos asimismo definir tipos más complejos usando patrones como:

```
<xs:element name="clave">

<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:pattern value="[a-zA-Z0-9]{8}"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

que nos define un elemento **clave** formado por ocho caracteres, sean éstos letras o números.

- **Restricciones a los espacios en blanco**

  Disponemos en XSD de una restricción que nos permite especificar cómo manejaremos los espacios en blanco. Se trata de la restricción

  whiteSpace. Por ejemplo:

```
<xs:element name="domicilio">

<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:whiteSpace value="preserve"/>
   </xs:restriction>
</xs:simpleType>

</xs:element>
```

nos permite definir un elemento **domicilio** donde especificamos al procesador de XML que no deseamos que elimine ningún espacio en blanco. Por otro lado, la siguiente definición:

ANOTACIONES

```
<xs:element name="domicilio">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:whiteSpace value="replace"/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

especifica al procesador de XML que deseamos que sustituya los caracteres *blancos* (tabuladores, saltos de línea, etc.) por espacios. Disponemos también de *collapse*, que reemplazará los caracteres *blancos* por espacios en blanco, pero que además reducirá múltiples espacios en blanco consecutivos a uno sólo y los caracteres al inicio de línea o al final de línea a uno sólo.

- **Restricciones de longitud**

Para restringir la longitud de un valor de un elemento disponemos de las restricciones: *length*, *maxLength* y *minLength*. Definiremos un elemento **clave**:

```
<xs:element name="clave">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:length value="8"/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

de tal manera que la longitud sea siempre 8. Podemos también definirlo para que tenga una longitud variable entre 5 y 8:

```
<xs:element name="clave">
<xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:minLength value="5"/>
   <xs:maxLength value="8"/>
   </xs:restriction>
</xs:simpleType>
</xs:element>
```

ANOTACIONES

En la siguiente tabla mostramos un resumen de las restricciones que podemos aplicar a los tipos de datos:

Tabla 8. Restricciones

Restricción	Descripción
enumeration	Define una lista de valores aceptables.
fractionDigits	Especifica el número máximo de dígitos decimales permitidos. Debe ser igual o superior a cero.
length	Especifica el tamaño exacto requerido. Debe ser igual o superior a cero.
maxExclusive	Especifica el límite superior para valores numéricos (el valor debe ser inferior a este número).
maxInclusive	Especifica el límite superior para valores numéricos (el valor debe ser inferior o igual a este número).
maxLength	Especifica el tamaño máximo permitido. Debe ser igual o superior a cero.
minExclusive	Especifica el límite inferior para valores numéricos (el valor debe ser superior a este número).
minInclusive	Especifica el límite inferior para valores numéricos (el valor debe ser superior o igual a este número).
minLength	Especifica el tamaño mínimo requerido. Debe ser igual o superior a cero.
pattern	Especifica el patrón que define la secuencia exacta de caracteres permitidos.
totalDigits	Especifica el número exacto de dígitos permitidos. Debe ser superior a cero.
whiteSpace	Especifica cómo deben tratarse los caracteres blanco (espacios, tabuladores, etc.).

Elementos complejos en XSD

Un elemento complejo es un elemento XML que contiene otros elementos y/o atributos. Podemos dividir los elementos complejos en cuatro clases principales:

– Elementos vacíos

– Elementos que contienen sólo otros elementos

– Elementos que contienen sólo texto

– Elementos que contienen otros elementos y texto

Todos estos elementos complejos también pueden contener atributos.

ANOTACIONES

Son elementos complejos de cada una de las clases los siguientes:

Un elemento **producto** vacío:

```
<producto id="1345"/>
```

Un elemento **alumno** que contiene otros elementos:

```
<alumno>
<nombre>Juan</nombre>
<apellido>Nadie</apellido>
</alumno>
```

Un elemento **alojamiento** que sólo contiene texto:

```
<alojamiento tipo="hotel">
Hostería Vientos del Sur
</alojamiento>
```

Un elemento **expedición** que contiene texto y elementos:

```
<expedicion destino="Fitz Roy">
Llegamos al Chaltén en la Patagonia el
<fecha>22.08.2003</fecha> ....
</expedicion>
```

- **Cómo definir un elemento complejo**

Si observamos el elemento **alumno** que contiene otros elementos:

```
<alumno>
<nombre>Juan</nombre>
<apellido>Nadie</apellido>
</alumno>
```

podemos definir este elemento XML de diferentes formas:

1. Podemos declarar directamente el elemento **alumno**:

```
<xs:element name="alumno">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="nombre" type="xs:string"/>
         <xs:element name="apellido" type="xs:string"/>
```

ANOTACIONES

```
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

De este modo, sólo el elemento **alumno** puede usar el tipo complejo definido. Observad que los elementos contenidos en **alumno** (**nombre** y **apellido**) están contenidos en un comando de tipo `sequence`, lo cual obliga a que en el elemento aparezcan en el mismo orden.

2. El elemento **alumno** puede tener un atributo de tipo que se refiera al tipo complejo que se vaya a usar:

```
<xs:element name="alumno" type="infopersona"/>

 <xs:complexType name="infopersona">
    <xs:sequence>
       <xs:element name="nombre" type="xs:string"/>
       <xs:element name="apellido" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Usando esta técnica, múltiples elementos se pueden referir al mismo tipo complejo, de esta manera:

```
<xs:element name="alumno" type="infopersona"/>
<xs:element name="profesor" type="infopersona"/>
<xs:element name="plantilla" type="infopersona"/>

<xs:complexType name="infopersona">
 <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Podemos también usar un tipo complejo como base para construir otros tipos complejos más elaborados:

```
<xs:element name="alumno" type="infopersona"/>

<xs:complexType name="persona">
 <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
 </xs:sequence>
```

ANOTACIONES

```
</xs:complexType>
<xs:complexType name="infopersona">
 <xs:complexContent>
   <xs:extension base="persona">
     <xs:sequence>
       <xs:element name="direccion" type="xs:string"/>
       <xs:element name="ciudad" type="xs:string"/>
       <xs:element name="país" type="xs:string"/>
     </xs:sequence>
   </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

- **Elementos vacíos**

Imaginemos el elemento XML siguiente:

```
<producto id="1345"/>
```

Se trata de un elemento XML que no contiene ni texto ni ningún elemento XML. Para definirlo, debemos definir un tipo que sólo permita elementos en su contenido y no declarar ningún elemento. Así:

```
<xs:element name="producto">
 <xs:complexType>
   <xs:complexContent>
     <xs:restriction base="xs:integer">
       <xs:attribute name="id" type="xs:positiveInteger"/>
     </xs:restriction>
   </xs:complexContent>
 </xs:complexType>
</xs:element>
```

Aquí definimos el tipo complejo que contiene sólo elementos (con `complexContent`). Esta directiva indica que deseamos derivar el contenido de un tipo complejo, pero no introducimos contenido. Además tenemos una restricción que añade un atributo entero. Podemos compactar la declaración:

```
<xs:element name="producto">
 <xs:complexType>
   <xs:attribute name="id" type="xs:positiveInteger"/>
 </xs:complexType>
</xs:element>
```

ANOTACIONES

O definirlo como un tipo con nombre para usarlo en la definición de otros elementos:

```
<xs:element name="producto" type="tipoproducto" />
<xs:complexType name="tipoproducto">
   <xs:attribute name="id" type="xs:positiveInteger"/>
</xs:complexType>
```

- **Definición de tipos complejos que contienen sólo elementos**

Observemos el siguiente elemento, que contiene sólo otros elementos:

```
<alumno>
<nombre>Juan</nombre>
<apellido>Nadie</apellido>
</alumno>
```

Podemos definir este elemento en XSD de la siguiente forma:

```
<xs:element name="alumno">
 <xs:complexType>
   <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>
```

Podemos, como en todos los casos anteriores, definirlo como un tipo con nombre para usarlo en múltiples elementos:

```
<xs:element name="alumno" tipo="persona">

 <xs:complexType name="persona">
   <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Observad el uso de `sequence`. Éste indica que los elementos (**nombre** y **apellido**) deben aparecer en este orden en el elemento.

ANOTACIONES

- Elementos que contengan sólo texto

Para definir elementos que sólo contengan texto, podemos definir una extensión o una restricción en un simpleContent, de la siguiente forma:

```
<xs:element name="elemento">
 <xs:complexType>
   <xs:simpleContent>
      <xs:extension base="tipo">
         ....
      </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

o como una restricción:

```
<xs:element name="elemento">
 <xs:complexType>
    <xs:simpleContent>
       <xs:restriction base="tipo">
          ....
       </xs:restriction>
    </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

Observemos el siguiente ejemplo de elemento que contiene sólo texto:

```
<alojamiento tipo="hotel">
Hostería Vientos del Sur
</alojamiento>
```

Una posible definición sería:

```
<xs:element name="alojamiento" type="tipoaloj"/>

<xs:complexType name="tipoaloj">
 <xs:simpleContent>
    <xs:extension base="xs:string">
       <xs:attribute name="tipo" type="xs:string" />
    </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

ANOTACIONES

- **Definición de tipos que contienen texto y elementos**

Estudiemos el siguiente elemento, que contiene texto y elementos:

```
<expedicion>
Llegamos al Chaltén en la Patagonia el
<fecha>22.08.2003</fecha> ....
</expedicion>
```

El elemento **fecha** es un elemento hijo de **expedicion**. Una posible definición sería:

```
<xs:element name="expedicion">
  <xs:complexType mixed="true">
       <xs:sequence>
          <xs:element name="fecha" type="xs:string"/>
       </xs:sequence>
  </xs:complexType>
</xs:element>
```

Para que en el contenido de **expedicion** además de elementos podamos incluir texto, debemos asignar el atributo `mixed` a **true**. Evidentemente, es posible definir el tipo con un nombre y usarlo para definir otros elementos o tipos.

Indicadores para tipos complejos

Los indicadores de XSD nos permiten controlar cómo se usarán los elementos en los tipos complejos. Disponemos de siete indicadores:

- Indicadores de orden:
  - All
  - Choice
  - Sequence

- Indicadores de ocurrencia:
  - maxOccurs
  - minOccurs

- Indicadores de grupo:
  - Group
  - attributeGroup

ANOTACIONES

- **Indicadores de orden**

Los indicadores de orden se utilizan para definir cómo ocurren los elementos.

– *Indicador* `All`

El indicador `all` especifica que los elementos hijos deben aparecer todos, sólo una vez cada uno, pero en cualquier orden.

```
<xs:complexType name="persona">
 <xs:all>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
 </xs:all>
</xs:complexType>
```

Si utilizamos este indicador, podemos usar el indicador `minOccurs` a 0 o a 1, y el `maxOccurs` sólo a 1 (estos dos indicadores están descritos más adelante).

– *Indicador* `Choice`

Este indicador especifica que sólo uno de los hijos puede aparecer:

```
<xs:complexType name="persona">
 <xs:choice>
    <xs:element name="documenIdent" type="xs:string"/>
    <xs:element name="pasaporte" type="xs:string"/>
 </xs:choice>
</xs:complexType>
```

– *Indicador* `Sequence`

Este indicador especifica que los hijos deben aparecer en un orden específico:

```
<xs:complexType name="persona">
 <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

ANOTACIONES

- **Indicadores de ocurrencia**

Los indicadores de ocurrencia permiten definir con qué frecuencia puede ocurrir un elemento.

Para todos los indicadores de orden y grupo (**any**, **all**, **choice**, **sequence**, **group**) el valor por defecto de `maxOccurs` y `minOccurs` es 1.

– *Indicador maxOccurs*

El indicador maxOccurs marcará el número máximo de veces que puede ocurrir un elemento.

Para que un elemento pueda aparecer un número ilimitado de veces, debemos asignar `maxOccurs="unbounded"`.

– *Indicador minOccurs*

Este indicador define el mínimo de veces que debe aparecer un elemento.

```
<xs:element name="alumno">
 <xs:complexType>
 <xs:sequence>
     <xs:element name="nombre" type="xs:string"/>
     <xs:element name="asignatura" type="xs:string"
          maxOccurs="10" minOccurs="0"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

Este ejemplo muestra cómo usar `minOccurs` y `maxOccurs` para limitar la aparición del elemento **asignatura** entre 0 y 10 veces.

- **Indicadores de grupo**

Los indicadores de grupo se utilizan para definir grupos de elementos relacionados.

– *Grupos de elementos*

ANOTACIONES

Los grupos de elementos se definen de la siguiente forma:

```
<xs:group name="nombregrupo">
 ...
</xs:group>
```

Dentro de la definición debemos usar un indicador de orden. Por ejemplo:

```
<xs:group name="grupoPersona">
 <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
    <xs:element name="apellido2" type="xs:string"/>
    <xs:element name="fechaNac" type="xs:date"/>
 </xs:sequence>
</xs:group>
```

Podemos usar este grupo para definir un tipo, un elemento, etc. Por ejemplo:

```
<xs:complexType name="infoAlumno">
 <xs:sequence>
   <xs:group ref="grupoPersona"/>
   <xs:element name="facultad" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

– *Grupos de atributos*

Los grupos de atributos tienen un comportamiento similar a los grupos de elementos. Se definen usando `attributeGroup` de la siguiente forma:

```
<xs:attributeGroup name="nombre">
 ...
</xs:attributeGroup>
```

Por ejemplo:

```
<xs:attributeGroup name="grupoAttPersona">
 <xs:attribute name="nombre" type="xs:string"/>
 <xs:attribute name="apellido" type="xs:string"/>
```

ANOTACIONES

```
  <xs:attribute name="apellido2" type="xs:string"/>
  <xs:attribute name="fechaNac" type="xs:date"/>
</xs:attributeGroup>
```

Usándolo luego de la siguiente forma:

```
<xs:element name="alumno">
  <xs:complexType>
     <xs:attributeGroup ref="grupoAttPersona"/>
  </xs:complexType>
</xs:element>
```

4.4. Transformaciones: XSL-T

XSL (*Extensible Stylesheet Language*, lenguaje de hojas de estilo extensible) es un lenguaje XML para expresar hojas de estilo (cómo debe representarse un lenguaje XML concreto). Consiste en tres componentes principales: XSLT (XSL *transformations*), XPath y XSL-FO (XSL-*formatting objects*).

A diferencia de HTML, donde el significado de cada etiqueta está claramente definido (salto de párrafo, salto de línea, cabecera, negrita) y donde resulta simple asignar estilos (fuentes, medidas, colores, etc.) a estas etiquetas, en XML las etiquetas no están definidas, sino que el usuario puede definirlas. En XML la etiqueta `tabla` puede representar una tabla de HTML en un caso y las medidas de un tablón de madera en otro, con lo cual los navegadores no saben cómo representar las etiquetas. Consecuentemente, el lenguaje de hojas de estilo para la presentación debe describir cómo mostrar un documento XML más claramente.

El lenguaje de estilos de XML, XSL, consiste en tres componentes principales:

- XSLT, un lenguaje de transformación de documentos.
- XPath, un lenguaje para referenciar a partes de documentos XML.
- XSL-FO, un lenguaje de formato de documentos XML.

ANOTACIONES

Con estos tres componentes XSL es capaz de:

- Transformar XML en XML, como por ejemplo XHTML o WML.
- Filtrar y/u ordenar datos XML.
- Definir partes de un documento XML.
- Formatear un documento XML sobre la base de los valores de los datos almacenados.
- Extraer los datos XML a XSL-FO que podremos usar para generar ficheros como PDF.

XSL es un lenguaje estándar del W3C. Fue estandarizado en dos etapas: la primera, en noviembre de 1999, incluía XSLT y XPath, mientras que la segunda, completada en octubre del 2000, incluía el XSL-FO.

XSLT es la parte del estándar XML que sirve para transformar documentos XML en otros documentos XML (como por ejemplo XHTML, WML, etc.).

Normalmente XSLT lo hace transformando cada elemento XML en otro elemento XML. XSLT también puede añadir otros elementos XML a la salida, o bien puede eliminar elementos. Asimismo puede reordenar o recolocar elementos y hacer comprobaciones y decisiones sobre qué elementos mostrar.

Durante la transformación, XSLT utiliza XPath para especificar o referenciar a partes del documento que cumplen uno o más patrones definidos. Cuando encuentra una coincidencia de patrones, XSLT transformará la parte coincidente del documento origen en el documento destino. Las partes no coincidentes no se transforman, sino que quedan en el documento destino sin ningún cambio.

4.4.1. Una transformación simple

Como casi todos los estándares del W3C, XSLT es en sí mismo un lenguaje XML y debe empezar con un elemento raíz. Dicho elemento raíz es del tipo `xsl:stylesheet` o `xsl:transform`

ANOTACIONES

(ambas etiquetas son totalmente equivalentes). La forma correcta de uso es:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Esta declaración identifica el espacio de nombres recomendado por el W3C. Si usamos este espacio de nombres, debemos incluir también el atributo `version` con el valor 1.0.

Declaración incorrecta

De acuerdo con los borradores del estándar, la declaración correcta de una hoja de estilo era:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

Dicha declaración está obsoleta, pero si el navegador utilizado es IE-5 debe usarse ésta.

El fichero XML de ejemplo que transformaremos es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<expediente alumno="Linus Torvalds">
 <asignatura id="1">
    <nombre>
    Programación básica
    </nombre>
    <nota>
    Notable
    </nota>
 </asignatura>
 <asignatura id="2">
    <nombre>
    Sistemas operativos
    </nombre>
    <nota>
    Excelente
    </nota>
 </asignatura>
 <asignatura>
    ...
```

ANOTACIONES

Este documento XML corresponde a un expediente de un estudiante de una universidad cualquiera. Se trata de un documento muy simple, pero totalmente válido para nuestras necesidades.

El documento XSL para convertir este documento XML en otro XHTML es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

  <html>
  <body>
    <h2>Expediente académico</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
       <th align="left">Asignatura</th>
       <th align="left">Nota</th>
    </tr>
    <xsl:for-each select="expediente/asignatura">
    <tr>
       <td><xsl:value-of select="nombre"/></td>
       <td><xsl:value-of select="nota"/></td>
    </tr>
    </xsl:for-each>
    </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Si llamamos al documento XSL `expediente.xsl`, podemos vincularlo con nuestro XML añadiendo una referencia a la hoja de estilo al principio del XML, del siguiente modo:

```
<?xml versión="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="expediente.xsl"?>
<expediente alumno="Linus Torvalds">
     <asignatura id="1">
        <nombre>
        Programación básica
        </nombre>
        <nota>
        Notable
        </nota>
     </asignatura>
     ...
```

ANOTACIONES

Si disponemos de un navegador con soporte XSL (como Mozilla en versiones superiores a la 1.2) para abrir el documento XML, el navegador utilizará el documento XSL para transformarlo en XHTML.

4.4.2. El elemento xsl: template

Una hoja de estilo de XSL consiste en una serie de plantillas (*templates*) de transformación. Cada elemento `xsl:template` contiene las transformaciones que XSL debe aplicar si el patrón especificado en el elemento coincide con lo encontrado en el documento XML.

Para especificar el elemento XML al que debemos aplicar el *template* utilizaremos el atributo `match` (también podemos aplicar la plantilla a todo el documento XML, para lo cual podemos especificar `match="/"`). Los valores que podemos asignar al atributo `match` son los especificados por el estándar XPath.

Por ejemplo, la siguiente transformación XSL devuelve un código XHTML concreto al procesar el documento con el expediente del alumno.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
 <html>
 <body>
    <h2>Expediente académico</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
       <th align="left">Asignatura</th>
       <th align="left">Nota</th>
    </tr>
    </table>
 </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Como podemos ver, si probamos este documento XSL, el resultado es tan sólo una cabecera de página. Si analizamos el documento

ANOTACIONES

XSL, veremos que dispone de una plantilla que se aplica cuando coincide con el elemento raíz del documento (`match="/"`) y que imprime al resultado lo que está contenido en la etiqueta.

4.4.3. El elemento value-of

El elemento `value-of` se usa para seleccionar y añadir a la salida el valor del elemento XML seleccionado.

Por ejemplo, si añadimos el siguiente código a nuestro ejemplo anterior:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
    <h2>Expediente académico</h2>
       <table border="1">
       <tr bgcolor="#9acd32">
          <th align="left">Asignatura</th>
          <th align="left">Nota</th>
       </tr>
       <tr>
          <td><xsl:value-of
             select="expediente/asignatura/nombre"/></td>
          <td><xsl:value-of
             select="expediente/asignatura/nota"/></td>
       </tr>
    </table>
   </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Veremos en el resultado que aparece la primera nota del expediente. Esto se debe a que las etiquetas `value-of` seleccionan el valor del primer elemento que cumple con el patrón especificado.

4.4.4. El elemento xsl: for-each

El elemento `xsl:for-each` de XSL puede utilizarse para seleccionar cada uno de los elementos del documento XML que pertenezcan a un conjunto determinado.

ANOTACIONES

Si al ejemplo anterior, donde sólo aparecía la primera nota del expediente, le añadimos un `xsl:for-each` que realice el recorrido por todo el expediente de la siguiente forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
     <h2>Expediente Académico</h2>
       <table border="1">
       <tr bgcolor="#9acd32">
          <th align="left">Asignatura</th>
          <th align="left">Nota</th>
       </tr>
         <xsl:for-each select="expediente/asignatura">
       <tr>
          <td><xsl:value-of select="nombre"/></td>
          <td><xsl:value-of select="nota"/></td>
       </tr>
       </xsl:for-each>
     </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Obtendremos ahora un listado de todas las notas de las asignaturas.

4.4.5. Ordenación de la información: xsl: sort

Para obtener una salida ordenada, simplemente debemos añadir un elemento `xsl:sort` al elemento `xsl:for-each` en nuestro fichero XSL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
     <h2>Expediente Académico</h2>
     <table border="1">
     <tr bgcolor="#9acd32">
         <th align="left">Asignatura</th>
         <th align="left">Nota</th>
       </tr>
         <xsl:for-each select="expediente/asignatura">
         <xsl:sort select="nombre"/>
```

ANOTACIONES

```
      <tr>
        <td><xsl:value-of select="nombre"/></td>
        <td><xsl:value-of select="nota"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

El atributo `select` nos permite indicar el elemento por el que vamos a realizar la ordenación, en este caso por nombre de asignatura.

4.4.6. Condiciones en XSL

Disponemos de dos elementos XSL que nos permiten implementar condiciones en nuestras transformaciones. Se trata de `xsl:if` y `xsl:choose`.

Elemento `xsl:if`

El elemento `xsl:if` nos permite aplicar una plantilla sólo en el caso de que la condición especificada se cumpla (sea cierta).

Un ejemplo de formato de `xsl:if` es el siguiente:

```
<xsl:if test="nota &lt; 5”>
 ..... sólo aparecerá con nota menor a 5 .....
</xsl:if>
```

Por ejemplo, podemos modificar el código anterior para que sólo muestre las notas superiores a 5.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>Expediente Académico</h2>
      <table border="1">
      <tr bgcolor="#9acd32">
        <th align="left">Asignatura</th>
```

ANOTACIONES

```
          <th align="left">Nota</th>
        </tr>
         <xsl:for-each select="expediente/asignatura">
         <xsl:if test="nota &gt; 5">
         <tr>
          <td><xsl:value-of select="nombre"/></td>
          <td><xsl:value-of select="nota"/></td>
        </tr>
        </xsl:if>
        </xsl:for-each>
      </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

El elemento `xsl:choose`

El elemento `xsl:choose` (junto con `xsl:when` y `xsl:otherwise`), nos permite modelar tests condicionales múltiples. Esto es, podemos, en función de una condición múltiple (con múltiples valores posibles), obtener resultados diversos.

Un ejemplo de formato de `xsl:choose` es el siguiente:

```
<xsl:choose>
 <xsl:when test="nota &lt; 5">
    ... código (suspendido) ....
 </xsl:when>
 <xsl:when test="nota &lt; 9">
    ... código (normal) ....
 </xsl:when>
 <xsl:otherwise>
    ... código (excelente) ....
 </xsl:otherwise>
</xsl:choose>
```

Modificamos el ejemplo anterior para que las notas inferiores a cinco aparezcan en color rojo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
     <h2>Expediente Académico</h2>
```

ANOTACIONES

```
      <table border="1">
      <tr bgcolor="#9acd32">
        <th align="left">Asignatura</th>
        <th align="left">Nota</th>
      </tr>
       <xsl:for-each select="expediente/asignatura">
       <tr>
        <td><xsl:value-of select="nombre"/></td>
        <td>
        <xsl:choose>
      <xsl:when test="nota &lt; 5">
        <font color="#FF0000">
         <xsl:value-of select="nota"/>
        </font>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="nota"/>
      </xsl:otherwise>
        </xsl:choose>
        </td>
        </tr>
        </xsl:if>
        </xsl:for-each>
     </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

4.4.7. El elemento xsl: apply-templates

`apply-templates` aplica una plantilla al elemento actual o a los elementos hijos del elemento actual. Con el atributo `select` podemos procesar sólo aquellos elementos hijos que especifiquemos y el orden en que deben procesarse.

Por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
 <html>
 <body>
   <h2>Expediente académico</h2>
   <xsl:apply-templates />
 </body>
 </html>
</xsl:template>
```

ANOTACIONES

```
<xsl:template match="expediente">
 <xsl:apply-templates select="asignatura"/>
</xsl:template>

<xsl:template match="asignatura">
<p>
 <xsl:apply-templates select="nombre"/>
 <xsl:apply-templates select="nota"/>
</p>
</xsl:template>

<xsl:template match="nombre">
Nombre: <xsl:value-of select="."/></span>
<br />
</xsl:template>

<xsl:template match="nota">
Nota: <xsl:value-of select="."/></span>
<br />
</xsl:template>

</xsl:stylesheet>
```

Como podemos ver, esta organización es mucho más modular y nos permite un mejor mantenimiento y revisión de la hoja de estilo.

4.4.8. Introducción a XPath

XPath es un estándar de W3C que define un conjunto de reglas para referenciar partes de un documento XML. De la definición del W3C podemos definir XPath como:

- XPath es una definición de una sintaxis para referenciar partes de un documento XML.
- XPath usa "caminos" o "rutas" para definir elementos XML.
- XPath define una librería de funciones.
- XPath es un elemento básico de XSL.
- XPath no está definido en XML.
- XPath es un estándar de W3C.

ANOTACIONES

XPath usa expresiones similares a las “rutas” o “caminos” a los ficheros en los sistemas operativos (por ejemplo, `/home/user/fichero.txt`).

Partiremos del siguiente fichero XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<expediente alumno="Linus Torvalds">
 <asignatura id="1">
    <nombre>
    Programación básica
    </nombre>
    <nota>
    Notable
    </nota>
 </asignatura>
 <asignatura id="2">
    <nombre>
    Sistemas operativos
    </nombre>
    <nota>
    Excelente
    </nota>
 </asignatura>
 <asignatura>
    ...
```

La siguiente expresión XPath selecciona el elemento **expediente** raíz:

```
/expediente
```

La siguiente selecciona todos los elementos **asignatura** del elemento **expediente**:

```
/expediente/asignatura
```

La siguiente selecciona todos los elementos **nota** de todos los elementos **asignatura** del elemento **expediente**:

```
/expediente/asignatura/nota
```

Nota

Cabe destacar que, al igual que en los sistemas de ficheros, si un elemento empieza con **/** indica una ruta absoluta a un elemento.

Si un elemento empieza con **//**, todos los elementos que cumplan el criterio serán seleccionados, independientemente del nivel del árbol XML donde se encuentren.

ANOTACIONES

Seleccionar elementos desconocidos

Al igual que ocurre con los sistemas de ficheros, podemos utilizar caracteres especiales (*) para indicar elementos desconocidos.

La siguiente expresión selecciona todos los elementos hijos de todos los elementos **asignatura** del elemento **expediente**:

```
/expediente/asignatura/*
```

La siguiente selecciona todos los elementos **nombre** nietos del elemento **expediente** independientemente del elemento padre:

```
/expediente/*/nombre
```

La siguiente expresión selecciona todos los elementos del documento:

```
//*
```

Seleccionar ramas del árbol

Podemos especificar qué partes del árbol de nodos queremos seleccionar usando corchetes ([ ]) en las expresiones XPath.

Podemos, por ejemplo, seleccionar el primer elemento **asignatura** del elemento **expediente**:

```
/expediente/asignatura[1]
```

Nota

Destacar que no existe una función first() en contraposición a last(). En su lugar hay que utilizar el marcador 1.

La siguiente expresión selecciona el último elemento **asignatura** hijo del elemento **expediente**:

```
/expediente/asignatura[last()]
```

La siguiente expresión selecciona todos los elementos **asignatura** hijos del elemento **expediente** que contengan un elemento **nota**:

```
/expediente/asignatura[nota]
```

La siguiente obliga, además, a que el elemento **nota** tenga un valor determinado:

```
/expediente/asignatura[nota>5]
```

La siguiente expresión selecciona los nombres de las asignaturas que tienen un elemento **nota** con valor concreto:

```
/expediente/asignatura[nota>5]/nombre
```

Seleccionar múltiples ramas

Podemos usar el operador | en las expresiones XPath para seleccionar múltiples caminos.

Por ejemplo, la expresión siguiente selecciona todos los elementos **nota** y **nombre** de los elementos **asignatura** del elemento **expediente**:

```
/expediente/asignatura/nombre/expediente/asignatura/nota
```

La siguiente selecciona todos los elementos **nombre** y **nota** en el documento:

```
//nombre //nota
```

Seleccionar atributos

En XPath los atributos se especifican mediante el prefijo @.

Esta expresión XPath selecciona todos los atributos llamados **id**:

```
//@id
```

La siguiente selecciona todos los elementos **asignatura** que tengan un atributo **id** con un valor concreto:

```
//asignatura[@id=1]
```

La siguiente selecciona todos los elementos **asignatura** que tengan algún atributo:

```
//asignatura[@*]
```

ANOTACIONES

Librería de funciones

XPath proporciona una librería de funciones que podemos utilizar en nuestros predicados XPath para afinar la selección, de la que forma parte la función `last()`vista anteriormente.

Algunas de las funciones más importantes son:

Tabla 9. Funciones

Nombre	Sintaxis	Descripción
count()	n=count(nodos)	Devuelve el número de nodos al conjunto proporcionado
id()	nodos=id(valor)	Selecciona nodos por su ID único
last()	n=last()	Devuelve la posición del último nodo en la lista de nodos que hay que procesar
name()	cad=name()	Devuelve el nombre del nodo
sum()	n=sum(nodos)	Devuelve el valor de la suma del conjunto de nodos especificado

Además, dispone de múltiples funciones de manipulación de cadenas, números, etc.

4.5. Práctica: creación de un documento XML, su correspondiente XML Schema y transformaciones con XSL-T

Vamos a crear un documento XML para almacenar información hotelera. Para ello diseñaremos en primer lugar el fichero XML Schema que vayamos a utilizar. Así podremos ir validando nuestro diseño a medida que añadamos partes.

El XML Schema resultante de nuestro diseño será:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Alojamientos">
    <xs:annotation>
```

ANOTACIONES

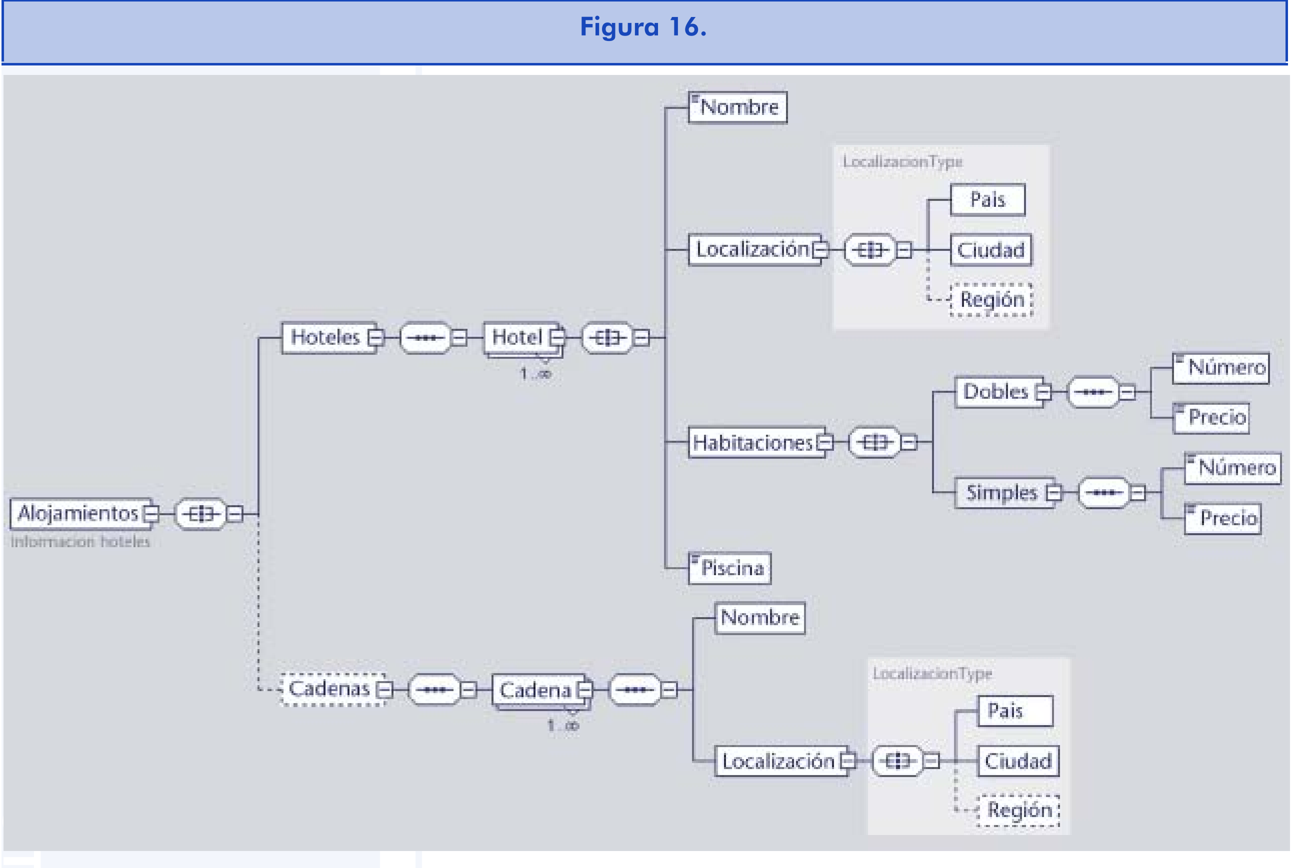
```
  <xs:documentation>Informacion hoteles</xs:documentation>
</xs:annotation>
<xs:complexType>
  <xs:all>
     <xs:element name="Hoteles">
       <xs:complexType>
         <xs:sequence>
            <xs:element name="Hotel" maxOccurs="unbounded">
              <xs:complexType>
                <xs:all>
                  <xs:element name="Nombre" type="xs:string"/>
                  <xs:element name="Localizacion"
                    type="LocalizacionType"/>
                  <xs:element name="Habitaciones">
                    <xs:complexType>
                      <xs:all>
                        <xs:element name="Dobles">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element
                                  name="Numero" type="xs:int"/>
                              <xs:element
                                name="Precio" type="xs:float"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="Simples">
                          <xs:complexType>
                           <xs:sequence>
                           <xs:element
                                  name="Numero" type="xs:int"/>
                           <xs:element
                              name="Precio" type="xs:float"/>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                    </xs:all>
                  </xs:complexType>
                </xs:element>
                <xs:element name="Piscina" type="xs:boolean"/>
                <xs:element name="Categoria" type="xs:int"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
     </xs:sequence>
     </xs:complexType>
    </xs:element>
   <xs:element name="Cadenas" minOccurs="0">
     <xs:complexType>
     <xs:sequence>
       <xs:element name="Cadena" maxOccurs="unbounded">
         <xs:complexType>
           <xs:sequence>
               <xs:element name="Nombre"/>
               <xs:element name="Localizacion"
                 type="LocalizacionType"/>
```

ANOTACIONES

```
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
       </xs:all>
     </xs:complexType>
   </xs:element>
   <xs:complexType name="LocalizacionType">
       <xs:all>
         <xs:element name="Pais"/>
         <xs:element name="Ciudad"/>
         <xs:element name="Region" minOccurs="0"/>
       </xs:all>
   </xs:complexType>
</xs:schema>
```

El diseño escogido para nuestro XML será:

Figura 16.



Creamos un documento a partir del esquema que hemos diseñado. Dicho documento, con datos de prueba será:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="hoteles.xslt"?>
<Alojamientos
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="hoteles.xsd">
   <Hoteles>
      <Hotel>
         <Nombre>Hotel Port Aventura</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Salou</Ciudad>
           <Region>Costa Dorada</Region>
         </Localizacion>
         <Habitaciones>
          <Dobles>
              <Numero>50</Numero>
              <Precio>133</Precio>
          </Dobles>
          <Simples>
             <Numero>10</Numero>
             <Precio>61</Precio>
          </Simples>
         </Habitaciones>
         <Piscina>false</Piscina>
         <Categoria>3</Categoria>
      </Hotel>
      <Hotel>
         <Nombre>Hotel Arts</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Barcelona</Ciudad>
         </Localizacion>
         <Habitaciones>
          <Dobles>
              <Numero>250</Numero>
              <Precio>750</Precio>
          </Dobles>
          <Simples>
              <Numero>50</Numero>
              <Precio>310</Precio>
          </Simples>
         </Habitaciones>
         <Piscina>true</Piscina>
         <Categoria>5</Categoria>
      </Hotel>
      <Hotel>
         <Nombre>Parador Seu d'Urgell</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Seu d'Urgell</Ciudad>
           <Region>Pirineo</Region>
         </Localizacion>
```

ANOTACIONES

```
          <Habitaciones>
            <Dobles>
               <Numero>40</Numero>
               <Precio>91.5</Precio>
            </Dobles>
            <Simples>
               <Numero>2</Numero>
               <Precio>44.4</Precio>
            </Simples>
          </Habitaciones>
          <Piscina>true</Piscina>
          <Categoria>4</Categoria>
       </Hotel>
   </Hoteles>
   <Cadenas>
      <Cadena>
         <Nombre>HUSA</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Barcelona</Ciudad>
         </Localizacion>
      </Cadena>
      <Cadena>
         <Nombre>NH Hoteles</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Pamplona</Ciudad>
         </Localizacion>
      </Cadena>
      <Cadena>
         <Nombre>Paradores de Turismo</Nombre>
         <Localizacion>
           <Pais>España</Pais>
           <Ciudad>Madrid</Ciudad>
         </Localizacion>
      </Cadena>
   </Cadenas>
</Alojamientos>
```

Utilizaremos ahora el siguiente documento XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0"
      encoding="UTF-8" indent="yes"/>
   <xsl:template match="Localizacion">
      <xsl:value-of select="Ciudad" />,
      <xsl:value-of select="Pais" />
      <xsl:if test="Region ">
         <i> (<xsl:value-of select="Region" />)</i>
      </xsl:if>
   </xsl:template>
   <xsl:template match="Hoteles">
      <h1>Lista de Hoteles</h1>
```

ANOTACIONES

```
    <xsl:for-each select=".">
      <xsl:apply-templates select="Hotel" />
    </xsl:for-each>
</xsl:template>

<xsl:template match="Piscina">
    <xsl:choose>
      <xsl:when test="node() = 'true'">
         <b>Si</b>
    </xsl:when>
      <xsl:otherwise>
               <b>No</b>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="Hotel">
    <h2>Hotel</h2>
    Nombre: <xsl:value-of select="Nombre" /> (Estrellas:
      <xsl:value-of select="Categoria" />)<br />
    Ubicación: <xsl:apply-templates select="Localizacion" /><br />
    Piscina: <xsl:apply-templates select="Piscina" /><br />
    <br />
    <h3>Habitaciones</h3>
    <table>
      <tbody>
        <tr>
          <th>Tipo</th>
          <th>Número</th>
          <th>Precio</th>
        </tr>
        <tr>
             <td>Simples</td>
             <td><xsl:value-of
                  select="Habitaciones/Simples/Numero" /></td>
             <td><xsl:value-of
                  select="Habitaciones/Simples/Precio" /></td>
        </tr>
        <tr>
             <td>Dobles</td>
             <td><xsl:value-of
                  select="Habitaciones/Dobles/Numero" /></td>
             <td><xsl:value-of
                  select="Habitaciones/Dobles/Precio" /></td>
        </tr>
      </tbody>
    </table>
</xsl:template>

<xsl:template match="Cadena">
    <h2>Cadena</h2>
    Nombre: <xsl:value-of select="Nombre" /> <br />
    Ubicación: <xsl:apply-templates select="Localizacion" /><br />
</xsl:template>

<xsl:template match="Cadenas">
```

```
      <h1>Lista de Cadenas hoteleras</h1>
      <xsl:for-each select=".">
         <xsl:apply-templates select="Cadena" />
      </xsl:for-each>
   </xsl:template>

   <xsl:template match="/">
      <html>
         <head>
         <title>Información hotelera</title>
         </head>
         <body>
            <xsl:apply-templates />
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Obtenemos un listado, en HTML, de los hoteles:

```
<?xml version="1.0" encoding="UTF-8"?> <html>
      <head>
        <title>Información hotelera</title>
      </head>
      <body><h1>Lista de Hoteles</h1><h2>Hotel</h2>
      Nombre: Hotel Port Aventura (Estrellas: 3)<br/>
      Ubicación: Salou,
      España<i> (Costa Dorada)</i><br/>
      Piscina: <b>No</b><br/><br/><h3>Habitaciones</h3><table>
         <tbody>
           <tr>
              <th>Tipo</th>
              <th>Número</th>
              <th>Precio</th>
           </tr>
           <tr>
              <td>Simples</td>
              <td>10</td>
              <td>61</td>
           </tr>
           <tr>
              <td>Dobles</td>
              <td>50</td>
              <td>133</td>
           </tr>
           </tbody>
         </table><h2>Hotel</h2>
      Nombre: Hotel Arts (Estrellas: 5)<br/>
      Ubicación: Barcelona,
      España<br/>
```

ANOTACIONES

```
      Piscina: <b>Si</b><br/><br/><h3>Habitaciones</h3><table>
        <tbody>
          <tr>
             <th>Tipo</th>
             <th>Número</th>
             <th>Precio</th>
          </tr>
          <tr>
             <td>Simples</td>
             <td>50</td>
             <td>310</td>
          </tr>
          <tr>
             <td>Dobles</td>
            <td>250</td>
             <td>750</td>
          </tr>
        </tbody>
        </table><h2>Hotel</h2>
      Nombre: Parador Seu d'Urgell (Estrellas: 4)<br/>
      Ubicación: Seu d'Urgell,
      España<i> (Pirineo)</i><br/>
      Piscina: <b>Si</b><br/><br/><h3>Habitaciones</h3><table>
        <tbody>
          <tr>
             <th>Tipo</th>
             <th>Número</th>
             <th>Precio</th>
          </tr>
          <tr>
             <td>Simples</td>
             <td>2</td>
             <td>44.4</td>
          </tr>
          <tr>
             <td>Dobles</td>
             <td>40</td>
             <td>91.5</td>
          </tr>
        </tbody>
        </table><h1>Lista de Cadenas hoteleras</h1><h2>Cadena</h2>
      Nombre: HUSA<br/>
      Ubicación: Barcelona,
      España<br/><h2>Cadena</h2>
      Nombre: NH Hoteles<br/>
      Ubicación: Pamplona,
      España<br/><h2>Cadena</h2>
      Nombre: Paradores de Turismo<br/>
      Ubicación: Madrid,
      España<br/></body>
</html>
```

ANOTACIONES

5. Contenido dinámico

5.1. CGI

Uno de los primeros mecanismos para generar contenido dinámico para la web es el API llamado CGI (acrónimo de *common gateway interface*, interfaz de pasarelas común). Éste es un mecanismo muy simple que permite que un servidor web ejecute un programa escrito en cualquier lenguaje de programación (ya sea como respuesta a un formulario HTML, a partir de un enlace, etc.), que le pueda pasar unos parámetros (bien provenientes del usuario, vía formularios, bien parámetros de configuración del servidor, del entorno de ejecución, etc.) y, finalmente, hace posible que el resultado de la ejecución de este programa sea enviado al usuario como una página web o cualquier otro tipo de contenido (un gráfico, etc.).

Gracias a este sencillo mecanismo, las páginas web, que hasta el momento de la aparición de CGI tenían unos contenidos estáticos e inmutables, son generadas dinámicamente en respuesta a peticiones concretas. Se abre así un nuevo mundo a los programadores de aplicaciones web. Estudiaremos a continuación el API de CGI, el cual ha quedado relegado en muchos casos a un papel secundario, pues adolece de muchos problemas, siendo la falta de rendimiento el más destacado de todos.

5.1.1. Introducción a los CGI

Al contrario de lo que sucede con los Servlets, etc., no existe ningún tipo de limitación al lenguaje de programación que podemos utilizar para escribir un CGI. Podemos usar desde *scripts* escritos en el lenguaje de la *shell* del sistema operativo hasta programas escritos en ensamblador, pasando por todo el abanico de lenguajes de programación existentes: C, C++, Perl, Python, etc. Hasta ahora, el lenguaje más popular para la escritura de CGI es Perl, ya que proporciona utilidades al programador que simplifican sobremanera la tarea de escribir programas CGI.

ANOTACIONES

5.1.2. Comunicación con los CGI

Lo primero que debemos recordar a la hora de escribir programas como CGI es el mecanismo de comunicación que nos proporciona el servidor web. Disponemos de dos opciones para enviar datos a un CGI (los datos generalmente procederán de un usuario, por norma general a partir de un formulario):

- Método GET. El método GET pasa toda la información (excepto ficheros) al CGI en la línea de dirección de la petición HTTP.

- Método POST. El método POST pasa toda la información al CGI en la entrada estándar, incluyendo ficheros.

Una vez recibe una petición que debe dirigir a un fichero CGI, el servidor ejecuta este programa, el CGI, y le envía la información a través de variables de entorno (o a través de la entrada estándar, si fuese pertinente). Algunas de las variables de entorno definidas por el estándar CGI son:

`SERVER_NAME` Nombre del servidor.

`SERVER_PROTOCOL` Protocolo utilizado por la petición.

`REQUEST_METHOD` Método utilizado para la invocación (GET o POST).

`PATH_INFO` Información de la ruta especificada en la petición.

`PATH_TRANSLATED` Ruta física a la ubicación del CGI en el servidor.

`SCRIPT_NAME` Nombre del CGI.

`REMOTE_ADDR` Dirección IP del ordenador que realiza la petición.

`REMOTE_HOST` Nombre del ordenador que realiza la petición.

`REMOTE_USER` Usuario que realiza la petición.

`AUTH_TYPE` Tipo de autentificación.

ANOTACIONES

CONTENT_TYPE Tipo MIME del contenido de la petición, especialmente útil en peticiones POST.

CONTENT_LENGTH Tamaño del contenido especialmente útil en peticiones POST.

La mayoría de servidores web proveen además de la variable de entorno llamada QUERY_STRING, que contiene los datos de la petición si ésta ha sido de tipo GET o si hemos añadido datos a la URL. Algunos servidores web añaden datos extra al entorno. La mayoría de estas variables adicionales empiezan con HTTP_ para evitar conflictos con versiones posteriores del estándar.

Ejemplo

Por ejemplo el servidor web Roxen añade una variable QUERY_ parámetro por cada parámetro de un formulario.

5.1.3. La respuesta de un CGI

Los CGI deben responder a las peticiones construyendo ellos mismos parte de la respuesta HTTP que recibirá el cliente. Es decir, deben indicar, en primer lugar, el tipo MIME del contenido que sirven. Pueden añadir posteriormente algunos campos adicionales (los especificados en el estándar HTTP). Después de una línea en blanco separadora, debe aparecer el contenido.

El CGI más simple posible, en este caso escrito en *shell script* y que enumera las variables de entorno comentadas anteriormente, es:

```
#!/bin/sh
echo Content-type: text/plain
echo
echo
echo SERVER_NAME=$SERVER_NAME
echo SERVER_PROTOCOL=$SERVER_PROTOCOL
echo REQUEST_METHOD=$REQUEST_METHOD
echo PATH_INFO=$PATH_INFO
echo PATH_TRANSLATED=$PATH_TRANSLATED
echo SCRIPT_NAME=$SCRIPT_NAME
echo REMOTE_ADDR=$REMOTE_ADDR
```

ANOTACIONES

```
echo REMOTE_HOST=$REMOTE_HOST
echo REMOTE_USER=$REMOTE_USER
echo AUTH_TYPE=$AUTH_TYPE
echo CONTENT_TYPE=$CONTENT_TYPE
echo CONTENT_LENGTH=$CONTENT_LENGTH
echo QUERY_STRING=$QUERY_STRING
```

Como podemos ver en este ejemplo (la sintaxis utilizada de *shell script* es muy simple), para listar las variables de entorno recibidas enviamos el tipo de contenido, seguido de la línea en blanco obligatoria y de todas y cada una de las variables de entorno mencionadas.

La ejecución de este servidor, sin parámetros adicionales, resulta en:



Figura 17.

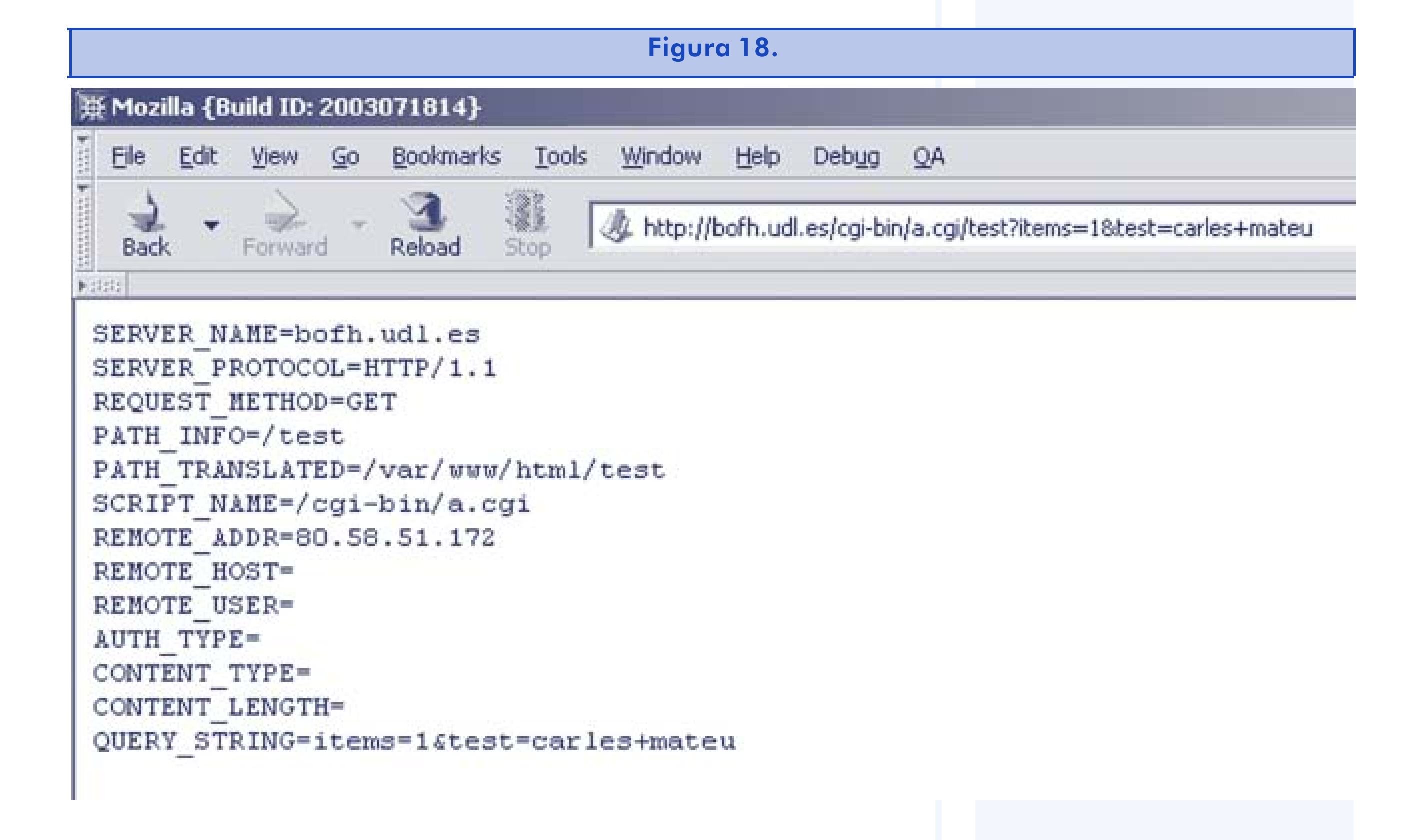



```
SERVER_NAME=bofh.udl.es
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
PATH_INFO=
PATH_TRANSLATED=
SCRIPT_NAME=/cgi-bin/a.cgi
REMOTE_ADDR=80.58.51.172
REMOTE_HOST=
REMOTE_USER=
AUTH_TYPE=
CONTENT_TYPE=
CONTENT_LENGTH=
QUERY_STRING=
```

Como podemos observar, si simplemente llamamos al CGI sin parámetros y no resulta su activación de un formulario, presenta pocas de las variables con valores. Si ahora llamamos al CGI pasándole

parámetros y un PATH extra (fijaos en el directorio que hay detrás del nombre del CGI), el resultado es el siguiente:

Figura 18.



Decodificación del QUERY_STRING

Como hemos podido observar en los ejemplos anteriores, los parámetros enviados a nuestro CGI seguían una codificación concreta y muy especial. Uno de los inconvenientes de usar CGI frente a alternativas más modernas, como Servlets, reside en el hecho de que debemos realizar la decodificación y análisis de dicha cadena manualmente. Afortunadamente existen librerías para casi todos los lenguajes de programación destinadas a facilitarnos la tarea.

Las reglas de codificación son las siguientes:

- Se separa la lista de parámetros del resto de la dirección URL con el carácter ?

- Se separan los parámetros (que siempre van en pares nombre, valor) mediante el carácter &. En algunos casos se acepta el carácter ; como sustituto para la separación.

ANOTACIONES

- Los nombres de parámetro se separan de los valores con el carácter =.

- Se sustituyen los caracteres especiales siguiendo la siguiente tabla:

  – El carácter ' ' (espacio en blanco) se convierte en +.

  – Los caracteres no alfanuméricos y los caracteres especiales, como los usados para codificación (+, etc.), se representan de la forma %HH, donde HH representa el valor hexadecimal del código ASCII del carácter.

  – Los saltos de línea se representan como %0D %0A.

5.1.4. Redirecciones

Podemos reenviar el cliente a una página diferente desde un programa CGI. Para ello sólo es necesario no devolver el código HTML estándar precedido del `Content-type`, sino que deberíamos devolver un campo de código de estado seguido de la localización de la nueva página como en el ejemplo:

```
#include <stdio.h>

int main()
 {
   printf("Status: 302\r\n");
   printf("Location: nueva.html\r\n");
   exit(1);
 }
```

5.2. PHP

PHP, cuyas siglas responden a un acrónimo recursivo (PHP: *hypertext preprocessor*), es un lenguaje sencillo, de sintaxis cómoda y similar a la de otros lenguajes como Perl, C y C++. Es rápido, interpretado, orientado a objetos y multiplataforma. Para él se encuentra disponible una multitud de librerías. PHP es un lenguaje ideal tanto para

ANOTACIONES

aprender a desarrollar aplicaciones web como para desarrollar aplicaciones web complejas. PHP añade a todo eso la ventaja de que el intérprete de PHP, los diversos módulos y gran cantidad de librerías desarrolladas para PHP son de código libre, con lo que el programador de PHP dispone de un impresionante arsenal de herramientas libres para desarrollar aplicaciones.

PHP suele ser utilizado conjuntamente con Perl, Apache, MySQL o PostgreSQL en sistemas Linux, formando una combinación barata (todos los componentes son de código libre), potente y versátil. Tal ha sido la expansión de esta combinación que incluso ha merecido conocerse con un nombre propio LAMP (formado por las iniciales de los diversos productos).

Apache, así como algunos otros servidores web, Roxen entre ellos, puede incorporar PHP como un módulo propio del servidor, lo cual permite que las aplicaciones escritas en PHP resulten mucho más rápidas que las aplicaciones CGI habituales.

5.2.1. Cómo funciona PHP

Si solicitamos a nuestro servidor una página PHP, éste envía dicha página al intérprete de PHP que la ejecuta (de hecho, no se trata más que de un programa) y devuelve el resultado (generalmente HTML) al servidor web, el cual, a su vez, se lo enviará al cliente.

Imaginemos que tenemos una página PHP con el siguiente contenido:

```
<?php echo "<h1>¡Hola mundo!</h1>";?>
```

Si tenemos este código en un fichero con extensión `.php` el servidor enviará la página al intérprete de PHP, el cual ejecuta la página y obtiene como resultado:

```
<h1>¡Hola mundo!</h1>
```

El servidor se lo enviará al navegador cliente que ha solicitado la página. El mensaje aparecerá en la pantalla de este último. Veremos que PHP permite mezclar en la misma página HTML y PHP, lo que

ANOTACIONES

facilita notablemente el trabajo con éste, pero por otro lado supone un peligro, ya que complica el trabajo en caso de que diseñadores de web y programadores trabajen conjuntamente en las páginas.

Disponemos, en los sistemas en los que esté instalado PHP, de un fichero de configuración global de PHP llamado `php.ini` que nos permitirá configurar algunos parámetros globales. Conviene revisar dicho fichero, pues aunque los valores por defecto suelen ser correctos, puede interesarnos realizar algunos cambios.

5.2.2. Sintaxis de PHP

Para empezar a comprender la sintaxis del lenguaje, analizaremos un programa mínimo de PHP:

```
<?php
 $MYVAR = "1234";
 $myvar = "4321";
 echo $MYVAR. "<br>\n";
 echo $myvar."<br>\n";
?>
```

La ejecución de este programa (su visualización desde un navegador), dará como resultado:

```
1234<br>
4321<br>
```

El primer punto que debemos destacar es que los bloques de código de PHP están delimitados en HTML con `<?php` y `?>`. Podemos, por tanto, escribir una página HTML e incluir en ella diversos bloques de instrucciones PHP:

```
<HTML>
 <HEAD>
    <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>
 <h1>Cabecera H1</h1>
 <?php echo "Hola" ?>
 <h1>Cabecera H1 segunda</h1>
```

ANOTACIONES

```
 <?php
 $MYVAR = 1234;
 $myvar = 4321;
 echo $MYVAR. "<br>";
 echo $myvar."<br>";
 // Este programa presenta en pantalla unos números
 ?>
 </BODY>
</HTML>
```

El siguiente punto que conviene destacar es que los nombres de variables se distinguen en que siempre deben empezar con $, y que igual que en C/C++, son *case sensitive*, es decir, diferencian mayúsculas y minúsculas. Fijaos también en que para concatenar texto (las variables y "
") utilizamos el carácter punto "." y además, en que todas las sentencias terminan con ";".

Asimismo conviene observar que las variables, a pesar de ser numéricas, se pueden concatenar con un texto ("
"). En este caso el intérprete convierte el valor numérico de la variable en texto para poder realizar la concatenación.

También podemos observar que hay un comentario dentro del código. Este comentario no afectará en modo alguno al programa ni será enviado al navegador del cliente (de hecho, el navegador cliente nunca recibirá código PHP). Para introducir comentarios en nuestro código, disponemos de dos opciones:

```
// Comentario de una sola línea

/*  Esto es un comentario de varias líneas.
    Para ello usamos este otro marcador
    de inicio y final de comentario */
```

5.2.3. Variables

PHP no precisa que declaremos *a priori* la variable que vamos a usar ni el tipo de ésta. PHP declarará la variable y le asignará el tipo de datos correcto en el momento en que la usemos por primera vez:

```
<?php $cadena = "Hola Mundo";
 $numero = 100;
 $decimal = 8.5;
?>
```

ANOTACIONES

ANOTACIONES

Como podemos observar, las tres variables fueron definidas en el momento de asignarles valor y no tuvimos que definir tipos.

En PHP las variables pueden tener, básicamente, dos ámbitos: uno global, en el que serán accesibles desde todo el código y otro local, en el que sólo serán accesibles desde la función en la que las creamos. Para asignar a una variable un ámbito global bastará con declararla (en este caso, sí que hace falta una declaración de variable) y usar la palabra reservada `global` en la declaración:

```
<?php
 global $test;
?>
```

Las variables que no califiquemos como globales, pero que sean definidas fuera de cualquier función, tendrán como ámbito el global.

Bastará con definir una variable dentro de una función. En ese caso, su ámbito quedará restringido a la función donde la declaremos.

```
<?php
 global $variable; // Variable global
 $a=1; // Variable global implícita
 function suma()
 {
    $b=1; // b es una variable local
    $res=$a+$b; // res es una variable local
 }
?>
```

Podemos ver que tanto `a` como `variable` son variables globales, mientras que `b` y `res` son variables locales.

Además, disponemos en PHP de variables de vectores o *arrays*. Éstas son variables que pueden contener listas de elementos, a los que accederemos por medio de un índice.

```
<?php
 $mares = array(); //con array() declaramos un vector
 $mares[0]= "Mediterráneo";
 $mares[1] = "Aral";
 $mares[2] = "Muerto";
?>
```

Como podemos ver, hemos declarado la variable `mares` con una llamada a `array()`. Esto indica a PHP que dicha variable es un vector de elementos.

Para acceder a los elementos individuales del vector, debemos utilizar el nombre del vector e indicar la posición del elemento al que queremos acceder entre corchetes. En PHP los vectores empiezan a numerarse en 0.

Además de vectores con índices numéricos, PHP soporta vectores cuyos índices sean cadenas de texto:

```
<?php
 $montañas=array(); //con array() declaramos un vector
 $montañas["Everest"]= "Himalaya";
 $montañas["Fitz Roy"] = "Andes";
 $montañas["Montblanc"] = "Alpes";

 echo $montañas["Everest"]; // Imprimirá Himalaya
?>
```

5.2.4. Operadores

Los operadores son símbolos que se utilizan para realizar tanto operaciones matemáticas como comparaciones u operaciones lógicas.

Los más habituales en PHP son:

- Operadores matemáticos:

  a) + Suma varios números: `5 + 4 = 9`.

  b) – Resta varios números: `5 - 4 = 1`.

  c) * Realiza una multiplicación: `3 * 3 = 9`.

  d) / Realiza una división: `10/2 = 5`.

  e) % Devuelve el residuo de una división: `10 % 3 = 1`.

  f) ++ Incrementa en 1: `$v++ (Incrementa $v en 1)`.

  g) — Decrementa en 1: `$v-- (Decrementa $v en 1)`.

ANOTACIONES

- Operadores de comparación:

  a) == Evalúa a cierto si la condición de igualdad se cumple:
  `2 == 2 (Verdadero).`

  b) != Evalúa a cierto si la condición de igualdad no se cumple
  `2 != 2 (Falso).`

  c) < Evalúa a cierto si un número es menor que el otro
  `2 < 5(Verdadero).`

  d) > Evalúa a cierto si un número es mayor que el otro
  `6 > 4 (Verdadero).`

  e) <=Evalúa a cierto si un número es menor o igual que otro
  `2 <= 5 (Verdadero).`

  f) >=Evalúa a cierto si un número es mayor o igual que otro
  `6 >= 4 (Verdadero).`

- Operadores lógicos:

  a) && Evalúa a cierto si los dos operadores son ciertos.

  b) || Evalúa a cierto si alguno de los operadores es cierto.

  c) `And` Evalúa a cierto si los operadores son ciertos.

  d) `Or` Evalúa a cierto si alguno de los operadores es cierto.

  e) `Xor` Evalúa a cierto si o un operador es cierto o lo es el otro.

  f) ! Invierte el valor de verdad del operador.

El siguiente ejemplo nos mostrará los operadores matemáticos más comunes:

```
<?php
 $a = 5;
 $b = 10;
 $c = ($a + $b); //$c vale 15
 $d = ($b - $a); //$d vale 5
 $e = ($a * $b); //$e vale 50
 $f = ($b / $a); //$f vale 2
 $g = ($b % $a); //$g vale 0
?>
```

5.2.5. Estructuras de control

Las estructuras de control de PHP nos permiten controlar el flujo de la operación de nuestro programa, controlando en todo momento qué porciones de códigos se ejecutan en función de determinadas condiciones.

Condicionales

Los condicionales son estructuras que permiten llevar a cabo determinadas operaciones sólo en caso de cumplirse una condición. Suelen llamarse también bifurcaciones, ya que permiten dividir el flujo de ejecución del programa en función del valor de verdad de una sentencia o condición.

Disponemos en PHP de dos condicionales principales, el condicional `if/else` y el condicional `switch`.

El condicional `if` nos permite escoger entre dos bloques de código en función del cumplimiento o no de una condición.

```
<?php
 $a = 0;
 $b = 1;
 if($a == $b)
 {
     echo "Pues resulta que 0 es igual a 1";
 }
 else
 {
     echo "Todo sigue igual. 0 no es igual a 1";
 }
?>
```

Si seguimos el flujo de ejecución de este programa, veremos que inicialmente creamos dos variables `a` y `b`, a las que asignamos dos valores numéricos diferentes. Acto seguido llegamos a la sentencia condicional `if`. Ésta verifica la veracidad o cumplimiento de la condición especificada. En este caso tenemos un operador `==` de igualdad, el cual devuelve que la comparación es falsa; por tanto, la sentencia `if` no ejecuta el primer bloque de código, el que habría

ANOTACIONES

ejecutado en caso de cumplirse la condición, sino que ejecuta el segundo, el precedido por `else`.

Podemos, pues, definir la estructura de `if/else` como:

```
if(condición)
 {
     código que se ejecutará si la condición es cierta
 }
else
 {
     código que se ejecutará si la condición es falsa
 }
```

Podemos comprobar más de una condición encadenando diversos `if/else`:

```
if(condición1)
 if(condición2)
    {
      código que se ejecutará si la condición2 es cierta
      y la condición 1 es cierta
    }
 else
    {
      código que se ejecutará si la condición2 es falsa
      y la condición 1 es cierta
    }
else
 {
    código que se ejecutará si la condición1 es falsa
 }
```

Un caso extendido del encadenamiento de `if/else` es el correspondiente a aquellos supuestos en los que debemos ejecutar código diferente en función del valor de una variable. A pesar de que podemos realizar un encadenamiento de `if/else` comprobando el valor de dicha variable, si debemos comprobar diversos valores el código resulta engorroso. Por ello PHP proporciona una construcción condicional más adecuada para ello denominada `switch`.

```
<?php
 $a=1;
 switch($a)
```

ANOTACIONES

```
{
   case 1:
   case 2: echo "A es 1 o 2"; break;
   case 3: echo "A es 3"; break;
   case 4: echo "A es 4"; break;
   case 5: echo "A es 5"; break;
   case 6: echo "A es 6"; break;
   default: echo "A es otro valor";
}
```

La ejecución de `switch` es un poco compleja; de hecho, es muy parecida a la de C. La sentencia `switch` se ejecuta línea a línea. Inicialmente no se ejecuta ningún código de ninguna línea. Cuando se encuentra un `case` con un valor que coincide con el de la variable del `switch` PHP empieza a ejecutar las sentencias. Éstas continúan siendo ejecutadas hasta el final de `switch` o hasta que se encuentre un `break`. Por eso en el ejemplo si la variable vale 1 o si la variable vale 2, se ejecuta el mismo bloque de código.

Disponemos, además, de un valor especial `default` que siempre coincide con el valor de la variable.

Bucles

La otra estructura de control importante es la de los bucles. Los bucles nos permiten ejecutar repetidamente un bloque de código en función de una condición.

Tenemos tres bucles principales en PHP: `for`, `while` y `foreach`.

- **El bucle `while`**

El bucle `while` es el más simple de los tres, pero aún así probablemente sea el bucle más utilizado. El bucle se ejecuta mientras la condición que le hemos pasado sea cierta:

```
<?php
 $a = 1;
 while($a < 4)
 {
    echo "a=$a<br>";
    $a++;
 }
?>
```

ANOTACIONES

En este caso, el bucle se ejecutará cuatro veces. Cada vez que se ejecute, incrementaremos el valor de `a` e imprimiremos un mensaje. Cada vez que se vuelve a ejecutar el código, `while` comprueba la condición y, en caso de cumplirse, vuelve a ejecutar el código. La cuarta vez que se ejecute, como `a` valdrá cuatro, no se cumplirá la condición especificada y el bucle no volverá a ejecutarse.

- **El bucle `for`**

Para bucles del tipo anterior, donde la condición de continuación es sobre una variable que aumenta o disminuye con cada iteración, disponemos de un tipo de bucle más apropiado: `for`.

El código anterior, usando `for` quedaría de la forma:

```
<?php
 for($a=1;$a < 4; $a++)
 {
    echo "a=$a<br>";
 }
?>
```

Como podemos ver, en el caso del bucle `for` en la misma sentencia declaramos la variable sobre la que iteraremos, la condición de finalización y la de incremento o continuación.

- **`foreach`**

Para aquellos casos en los que queremos que nuestro bucle haga un recorrido sobre los elementos de un vector disponemos de una sentencia que nos simplifica este hecho: `foreach`.

```
<?php
$a = array (1, 2, 3, 17);
foreach ($a as $v
{
 print "Valor: $v.\n";
}
?>
```

ANOTACIONES

Como podemos ver, en su forma más simple, `foreach` asigna a una variable `v` uno a uno todos los valores de un vector a.

5.2.6. Funciones

Otro punto clave de PHP son las funciones. Las funciones en PHP pueden recibir o no, parámetros y siempre pueden devolver un valor. Las funciones sirven para dar mayor modularidad al código, evitando así la repetición de código, permitiendo el aprovechamiento de código entre proyectos, etc.

Un esquema de función es el siguiente:

```
<?php
 function ejemp ($arg_1, $arg_2, ..., $arg_n)
 {
 // Código de la función
 return $retorno;
 }
?>
```

Podemos llamar a las funciones desde el código principal o desde otras funciones:

```
<?php
      function suma ($a1, $a2)
      {
         $retorno=$a1+$a2;
         return $retorno;
      }

      function sumatorio ($b1, $b2, $b3)
      {
         for ($i=$b1;$i<$b2;$i++)
         {
           $res=suma($res,$b3);
         }
         return $res;
      }
      echo sumatorio(1,3,2);
?>
```

ANOTACIONES

El resultado de ejecutar dicho programa será que imprimirá un número seis.

Las funciones en PHP reciben habitualmente los parámetros por valor, es decir, la variable que se pasa como parámetro en el código que llama no sufre modificaciones si el parámetro de la función es modificado. Podemos, no obstante, pasar parámetros por referencia (de forma similar a los punteros de otros lenguajes de programación):

```
<?php
 function modifi ($&a1, $a2)
 {
     $a1=0;
     $a2=0;
 }
 $b1=1;
 $b2=1;
 modifi($b1,$b2);
 echo $b1." ".$b2;
?>
```

En este caso, el resultado del programa será:

```
1 0
```

5.2.7. Uso de PHP para aplicaciones web

Para usar PHP como lenguaje de desarrollo de aplicaciones web, la primera necesidad que tenemos es saber cómo interactuará PHP con nuestro usuario web. Podemos dividir dicha interactuación en dos partes, mostrando información al usuario y recogiendo información de éste.

Mostrando información

Tenemos dos mecanismos para que PHP muestre información al usuario: por un lado podemos escribir páginas HTML corrientes, insertando sólo el código PHP que requerimos en medio del código HTML. Por ejemplo:

ANOTACIONES

```
<HTML>
 <HEAD>
     <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>
 <h1>Cabecera H1</h1>
 <?php $a=1; ?>
 <h1>Cabecera H1 segunda</h1>
 <?php $b=1; ?>
 </BODY>
</HTML>
```

Por otro lado, podemos usar PHP para generar contenido dinámico. Para ello debemos usar las instrucciones de PHP de salida de datos, la más importante, `echo`.

```
<HTML>
 <HEAD>
     <TITLE>Título del documento</TITLE>
 </HEAD>
<BODY>
 <h1>Cabecera H1</h1>
 <?php echo "Contenido de la <B>página</B>"; ?>
 <h1>Cabecera H1 segunda</h1>
 </BODY>
</HTML>
```

Recogida de información del usuario

Para recoger información del usuario, podemos utilizar los formularios de HTML, utilizando nuestros programas PHP como `ACTION` de dichos formularios. Como PHP fue diseñado para crear aplicaciones web, el acceso a los valores introducidos por el usuario en los campos del formulario es realmente fácil en PHP, ya que éste define un vector llamado `REQUEST` al que accedemos con el nombre del campo como índice y que contiene el valor contenido en éste al ejecutar el programa PHP.

ANOTACIONES

Si tenemos este formulario:

```
<HTML>
   <HEAD>
      <TITLE>Título del documento</TITLE>
   </HEAD>
<BODY>
   <FORM ACTION="programa.php" METHOD=GET>
   Entre su nombre: <INPUT TYPE=TEXT NAME="nombre">
   <INPUT TYPE=submit>
   </FORM>
   </BODY>
</HTML>
```

Y definimos el siguiente programa PHP como `programa.php` para que responda al formulario:

```
<HTML>
  <HEAD>
      <TITLE>Título del documento</TITLE>
  </HEAD>
  <BODY>
  <?php
  echo "Hola ".$REQUEST["nombre"];
  ?>
  </BODY>
</HTML>
```

Este programa recogerá el nombre introducido por el usuario y nos lo mostrará por pantalla.

5.2.8. Funciones de cadena

PHP provee de un conjunto de funciones muy interesantes para el trabajo con cadenas de texto. Algunas de las más destacadas son:

`strlen` Devuelve la longitud de una cadena.

`explode` Divide una cadena en función de un carácter separador, y devuelve un vector con cada uno de los trozos de la cadena.

ANOTACIONES

`implode` Actúa al revés que `explode`, uniendo diversas cadenas de un vector con un carácter de unión.

`strcmp` Compara dos cadenas a nivel binario.

`strtolower` Convierte una cadena a minúsculas.

`strtoupper` Convierte una cadena a mayúsculas.

`chop` Elimina el último carácter de una cadena, útil para eliminar saltos de línea o espacios finales superfluos.

`strpos` Busca dentro de una cadena otra cadena especificada y devuelve su posición.

`str_replace` Reemplaza en una cadena una aparición de una subcadena por otra subcadena.

Podemos ver el funcionamiento de algunas de estas funciones en el siguiente ejemplo:

```
<?php
 $cadena1 = "hola";
 $cadena2 = "pera,manzana,fresa";
 $longitud = str_len($cadena1); //longitud=4
 $partes = explode(",",$cadena2);
 //genera el array $partes con $partes[0]="pera",
 //$partes[1]="manzana"; y $partes[2]="fresa";
 $chop = chop($cadena); // chop elimina la "a"
 $cadena3 = str_replace(",",";",$otracadena);
 //$cadena3 contiene: pera-manzana-fresa
 //Cambiamos las , por -
?>
```

5.2.9. Acceso a ficheros

PHP proporciona un amplio repertorio de métodos para el acceso a ficheros. Vamos a mostrar la más práctica y simple, muy adecuada si los ficheros a los que accederemos son pequeños.

ANOTACIONES

El código que comentaremos es el siguiente:

```
<?php
 $fichero = file("entrada.txt");
 $numlin = count($fichero);
 for($i=0; $i < $numlin; $i++)
 {
     echo $fichero[$i];
 }
?>
```

En este ejemplo leemos un archivo que tiene por nombre `entrada.txt` y lo mostramos como salida. El primer paso que seguimos es declarar la variable `fichero`, lo que nos genera un vector en el que PHP colocará todas las líneas del archivo. Para ello usaremos la función de librería `file`. El siguiente paso consiste en averiguar cuántos elementos contiene `fichero`. Para ello usamos la función `count`, que nos devuelve el tamaño de un vector, en este caso el vector que hemos generado al leer el fichero. Finalmente podemos escribir un bucle que recorrerá el vector tratando cada una de las líneas del archivo.

PHP proporciona muchas funciones más de tratamiento de ficheros. Disponemos, por ejemplo, de la función `fopen`, que permite abrir ficheros o recursos sin leerlos íntegramente en memoria y que es capaz de abrir ficheros de la siguiente forma:

```
 <?php
 $recurso = fopen ("entrada.txt", "r");
 $recurso = fopen ("salida.gif", "wb");
 $recurso = fopen ("http://www.uoc.edu/", "r");
 $recurso = fopen (
    "ftp://usuario:password@uoc.edu/salida.txt", "w");
?>
```

Ahí podemos ver cómo abrimos un fichero para lectura (``r''), para escritura binaria (``wb''), una página web para leerla como si se tratara de un fichero y un fichero vía FTP para escribirlo, respectivamente.

5.2.10. Acceso a bases de datos

PHP proporciona métodos para acceder a un gran número de sistemas de bases de datos (mySQL, PostgreSQL, Oracle, ODBC, etc.).

ANOTACIONES

Esta funcionalidad es imprescindible para el desarrollo de aplicaciones web complejas.

Acceso a mySQL desde PHP

mySQL es uno de los sistemas de bases de datos más populares en el desarrollo de aplicaciones web ligeras por su alto rendimiento para trabajar con bases de datos sencillas. Gran cantidad de aplicaciones web de consulta, etc. están desarrolladas con el dueto PHP-mySQL. Por eso el API de acceso a mySQL de PHP está altamente desarrollado.

Vamos a ver un ejemplo de acceso a la base de datos desde PHP para comprobar así la sencillez con la que podemos usar bases de datos en nuestras aplicaciones web:

```
<?php
 $conexion=mysql_connect($servidor,$usuario,$password);
 if(!$conexion).
 {
    exit();
 }
 if(!(mysql_select_db($basedatos,$conexion)))
 {
    exit();
 }
 $consulta=mysql_query(
    "select nombre, telefono from agenda order by nombre",
    $conexion);
 while($fila = mysql_fetch_array($consulta))
 {
    $nombre = $fila["nombre"];
    $teléfono = $fila["telefono"];
    echo "$nombre: $telefono\n<br>";
 }
 mysql_free_result($consulta);
 mysql_close($conexion);
?>
```

Podemos ver que el primer paso para acceder a la base de datos es abrir una conexión con ésta. Para ello necesitaremos la dirección del ordenador que contenga la base de datos, el usuario con el que co-

ANOTACIONES

nectaremos y la palabra de acceso a dicha base de datos. Una vez conectados al servidor de mySQL, debemos seleccionar qué base de datos de las múltiples que puede tener el servidor queremos usar para trabajar. A raíz de esta secuencia de conexión, tendremos en la variable conexión los datos de la conexión a mySQL. Debemos pasar esta variable a todas las funciones de PHP que accedan a bases de datos. Ello nos permite disponer de diversas conexiones a diferentes bases de datos abiertas al mismo tiempo y trabajar con ellas simultáneamente.

El siguiente paso será ejecutar una sentencia de consulta de bases de datos en el lenguaje de nuestra base de datos, en este caso SQL. Para ello usaremos la función de PHP `mysql_query`, que nos devolverá el resultado de la consulta y que guardaremos en la variable `consulta`. La consulta concreta enumera el contenido de una tabla de la base de datos llamada `agenda`, la cual contiene dos columnas denominadas `nombre` y `telefono`.

Posteriormente, podemos ejecutar un bucle que recorrerá todos los registros para que nos devuelva la consulta a la base de datos, accediendo a ellos uno a uno y pudiendo así mostrar los resultados.

Una vez hayamos acabado el acceso a la base de datos, debemos liberar la memoria y los recursos consumidos con la consulta. Para ello utilizaremos la función `mysql_free_result` y luego podremos cerrar la conexión a mySQL.

Acceso a PostgreSQL desde PHP

De forma similar a como accedemos a mySQL, podemos acceder a bases de datos que tengamos en servidores PostgreSQL. Para ello utilizaremos, al igual que en la sección anterior, un código que enumerará el contenido de nuestra tabla `agenda`.

```
<?php
 //conectamos a la base de datos
 //$conexion = pg_connect("dbname=".$basedatos);
 // conectamos a la base de datos en servidor puerto "5432"
 //$conexion = pg_connect(
// "host=$servidor port=5432 dbname=$basedatos");
```

ANOTACIONES

```
   //conectamos a la base de datos en servidor puerto "5432"
   //con usuario y password
   $conexion = pg_connect("host=$servidor port=5432 ".
      "dbname=$basedatos user=$usuario password=$password")
      or die "No conecta";

   $resultado = pg_query($conexión,
      "select nombre, telefono from agenda order by nombre");

   while($fila = pg_fetch_array($result))
  {
      $nombre = $fila["nombre"];
      $telefono = $fila["telefono"];
      echo "$nombre: $telefono\n<br>";
  }
  pg_close($dbconn);

?>
```

Como podemos observar si comparamos este ejemplo con el anterior realizado en mySQL, existe gran similitud entre ambos códigos. No obstante, a pesar de las similitudes, el API de PostgreSQL para PHP presenta algunas diferencias respecto al de mySQL, ya que por un lado ofrece mayor flexibilidad a la hora de conectar y por otro lado, proporciona el soporte para trabajar con objetos grandes, etc., lo cual muestra la mayor potencia de PostgreSQL frente a mySQL. Un punto controvertido en el API de PostgreSQL es que nos aísla totalmente del sistema de transacciones de PostgreSQL, cosa que a pesar de que en la mayoría de situaciones es más que correcta, existen casos en los que sería deseable un mayor control sobre éstas.

5.2.11. Para seguir profundizando

Uno de los puntos fuertes y una de las claves del éxito de PHP como lenguaje de programación de aplicaciones web reside en la gran cantidad de librerías, módulos, etc., que se han desarrollado para él. PHP pone a nuestra disposición una cantidad ingente de API, funciones, módulos, clases (recordad que, progresivamente, PHP se está convirtiendo en un lenguaje de programación orientado a objetos), los cuales nos permiten operar con la complejidad creciente de las aplicaciones web. Esta diversidad de soporte incluye, entre otras cosas:

- Control de sesiones.
- Control de identidad de usuarios.

ANOTACIONES

- Plantillas HTML.
- Carritos de compra.
- Creación dinámica de HTML.
- Creación de imágenes dinámicamente.
- Manejo de *cookies.*
- Transferencia de ficheros.
- Manejo de XML, XSLT, etc.
- Múltiples protocolos de comunicaciones: HTTP, FTP, etc.
- Creación de ficheros PDF.
- Acceso a directorios LDAP.
- Interfaces a multitud de bases de datos: Oracle, Sybase, etc.
- Expresiones regulares.
- Manejo de equipos de red: SNMP.
- Servicios web: XMLRPC, SOAP.
- Manejo de contenidos Flash.

Además de su utilidad como lenguaje de programación de aplicaciones web, PHP está siendo cada vez más usado como lenguaje de programación de propósito general, incluyendo en su arsenal de funciones interfaces hacia las librerías de interfaces gráficos más conocidas (Win32 o GTK, entre otras), acceso directo a funciones del sistema operativo, etc.

Por eso, en caso de desear usar PHP para desarrollar algún proyecto, resulta altamente recomendable visitar la página web del proyecto (http://www.php.net), ya que es muy posible que nos ofrezca un surtido de herramientas que nos faciliten grandemente el trabajo. Disponemos, además, de PEAR, un repositorio de PHP que nos proporcionará la mayor parte de las herramientas que podamos llegar a necesitar.

ANOTACIONES

5.3. Java Servlets y JSP

5.3.1. Introducción a los Java Servlets

Los Servlets de Java son la propuesta de la tecnología Java para el desarrollo de aplicaciones web. Un Servlet es un programa que se ejecuta en un servidor web y que construye una página web que es devuelta al usuario. Esta página, construida dinámicamente, puede contener información procedente de bases de datos, ser una respuesta a los datos introducidos por el usuario, etc.

Los Servlets Java presentan una serie de ventajas sobre los CGI, el método tradicional de desarrollo de aplicaciones web. Éstos son más portables, más potentes, mucho más eficientes, más fáciles de usar, más escalables, etc.

Eficiencia

Con el modelo tradicional de CGI, cada petición que llega al servidor dispara la ejecución de un nuevo proceso. Si el tiempo de vida del CGI (el tiempo que tarda en ejecutarse) es corto, el tiempo de instanciación (el tiempo de arrancar un proceso) puede superar al de ejecución. Con el modelo de Servlets, la máquina virtual de Java, el entorno donde se ejecutan, se arranca al iniciar el servidor, permaneciendo arrancada durante toda la ejecución del mismo. Para atender cada petición no se arranca un nuevo proceso, sino un *thread*, un proceso ligero de Java, mucho más rápido (de hecho, casi instantáneo). Además, si tenemos *x* peticiones simultáneas de un CGI, tendremos *x* procesos simultáneos en memoria, consumiendo así *x* veces el espacio de un CGI (que, en caso de ser interpretado, como suele ocurrir, implica el consumo de *x* veces el intérprete). En el caso de los Servlets, hay determinada cantidad de *threads*, pero sólo una copia de la máquina virtual y sus clases.

El estándar de Servlets también nos ofrece más alternativas que los CGI para optimizaciones: `caches` de cálculos previos, `pools` de conexiones de bases de datos, etc.

ANOTACIONES

Facilidad de uso

El estándar de Servlets nos ofrece una magnífica infraestructura de desarrollo de aplicaciones web, proporcionándonos métodos para análisis automático y decodificación de los datos de los formularios de HTML, acceso a las cabeceras de las peticiones HTTP, manejo de `cookies`, seguimiento, control y gestión de sesiones, entre otras muchas facilidades.

Potencia

Los Servlets Java permiten hacer muchas cosas que son difíciles o imposibles de realizar con los CGI tradicionales. Los Servlets pueden compartir los datos entre sí, permitiendo compartir datos, conexiones a bases de datos, etc. Asimismo, pueden mantener información de solicitud en solicitud, facilitando tareas como el seguimiento de las sesiones de usuario, etc.

Portabilidad

Los Servlets están escritos en Java y se rigen por un API estándar bien documentado. Como consecuencia de ello, los Servlets pueden ejecutarse en todas las plataformas que nos ofrezcan soporte de Java Servlets, sin tener que recompilar, modificarse, etc., sean estas plataformas Apache, iPlanet, IIS, etc., y además, con independencia de sistema operativo, arquitectura *hardware*, etc.

5.3.2. Introducción a las Java Server Pages o JSP

Las Java Server Pages (JSP) son una tecnología que nos permite mezclar HTML estático con HTML generado dinámicamente mediante código Java embebido en las páginas. Cuando programamos aplicaciones web con CGI, gran parte de la página que generan los CGI es estática y no varía de ejecución en ejecución. Por su parte, la parte variable de la página es realmente dinámica y muy pequeña. Tanto los CGI como los Servlet nos obligan a generar la página por completo desde nuestro código de programa, dificultando así las tareas de mantenimiento, diseño gráfico, comprensión del código, etc. Los JSP, por otro lado, nos permiten crear las páginas fácilmente.

Ejemplo

Las partes de la página que no varían de ejecución a ejecución son, las cabeceras, los menús, las decoraciones, etc.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 <HTML>
    <HEAD>
       <TITLE>Tienda. Bienvenido.</TITLE>
    </HEAD>
 <BODY>
    <H1>Bienvenido a la tienda</H1>
    <SMALL>Welcome,
    < % out.println(Utiles.leerNombreDeCookie(request)); %>
    </SMALL>
 </BODY>
</HTML>
```

Como podemos ver en el ejemplo, una página JSP no es más que una página HTML donde, merced a unas marcas especiales `< %` y `%>`, podemos incluir código Java.

Esto presenta una serie de ventajas obvias: por un lado disponemos de prácticamente las mismas ventajas que al usar Java Servlets; de hecho, los servidores JSP "traducen" éstos a Servlets antes de ejecutarlos. Por otro lado, los JSP nos aportan una simplicidad y una facilidad de desarrollo sustanciales. Resulta mucho más fácil escribir la página del ejemplo que escribir un Servlet o un CGI que imprima cada una de las líneas de la página anterior.

No obstante, esta simplicidad es también el inconveniente que presentan los JSP. Si deseamos realizar una aplicación compleja, con multitud de cálculos, accesos a bases de datos, etc., la sintaxis de los JSP, embebida en medio del HTML se torna farragosa. Por ello los JSP y los Servlets no suelen competir, sino que se complementan ya que, por otro lado, los estándares incluyen facilidades de comunicación entre ellos.

5.3.3. El servidor de Servlets/JSP

Para poder utilizar tanto Servlets como JSP en nuestro servidor web, debemos, por norma general, complementar éste con un servidor de Servlets/JSP (comúnmente denominado contenedor de Servlets). Disponemos de multitud de contenedores de código libre y de comer-

ANOTACIONES

ciales. Sun, la inventora de Java, mantiene una lista actualizada de contenedores de Servlets en:

http://java.sun.com/products/servlet/industry.html

- Apache Tomcat. Tomcat es la implementación de referencia oficial para las especificaciones Servlet y JSP a partir de las versiones (2.2 y 1.1 respectivamente). Tomcat es un producto muy robusto, altamente eficiente y uno de los más potentes contenedores de Servlets existentes. Su único punto débil reside en la complejidad de su configuración, dado el gran número de opciones existente. Para más detalles podemos acceder a la página oficial de Tomcat: http://jakarta.apache.org/.

- JavaServer Web Development Kit (JSWDK). El JSWDK era la implementación de referencia oficial para las especificaciones Servlet 2.1 y JSP 1.0. Se usaba como pequeño servidor para probar Servlets y páginas JSP en desarrollo. Actualmente ha sido abandonado en favor de Tomcat. Su página está en: http://java.sun.com/products/servlet/download.html.

- Enhydra. Enhydra es un servidor de aplicaciones que incluye, entre muchas otras funcionalidades, un contenedor de Servlets/JSP muy potente. Hendirá (http://www.enhydra.org) constituye una herramienta muy poderosa para desarrollar servicios y aplicaciones web, incluyendo en su conjunto de herramientas control de bases de datos, plantillas, etc.

- Jetty. Se trata de un servidor web/contenedor de Servlets muy ligero, escrito íntegramente en Java que soporta las especificaciones Servlet 2.3 y JSP 1.2. Es un servidor muy adecuado para desarrollo por su pequeño tamaño y su pequeña ocupación de memoria. Encontraremos su página web en: http://jetty.mortbay.org/jetty/index.html.

5.3.4. Un Servlet sencillo

El siguiente ejemplo nos muestra la estructura básica de un Servlet básico que maneja peticiones GET de HTTP (los Servlets también pueden manejar peticiones POST).

ANOTACIONES

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletBasico extends HttpServlet
{
 public void doGet(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {

    // Disponemos de request para acceder a los datos de la
    // petición HTTP.
    // Disponemos de response para modificar la respuesta HTTP
    // que generara el Servlet.

    PrintWriter out = response.getWriter();
    // Podemos usar out para devolver datos al usuario
    out.println("¡Hola!\n");
 }
}
```

Para escribir un Servlet debemos escribir una clase de Java que extienda (por herencia) la clase `HttpServlet` (o la clase más genérica `Servlet`) y que sobreescriba el método `service` o alguno de los métodos de petición más específicos (`doGet`, `doPost`, etc.).

Los métodos de servicio (`service`, `doPost`, `doGet`, etc.) tienen dos argumentos: un `HttpServletRequest` y un `HttpServletResponse`.

El `HttpServletRequest` proporciona métodos que nos permiten leer la información entrante como los datos de un formulario de HTML (FORM), las cabeceras de la petición HTTP, los *cookies* de la petición, etc. Por otro lado, el `HttpServletResponse` tiene métodos que nos permiten especificar los códigos de respuesta de HTTP (200, 404, etc.), las cabeceras de respuesta (`Content-Type`, `Set-Cookie`, etc.). Y lo más importante, nos permiten obtener un `PrintWriter` (una clase de Java que representa un "fichero" de salida) usado para generar los datos de salida que se enviarán de vuelta al cliente. Para Servlets sencillos, la mayoría del código se destina a trabajar con este `PrintWriter` en sentencias `println` que generan la página deseada.

ANOTACIONES

5.3.5. Compilación y ejecución de los Servlets

El proceso de compilación de los Servlets es muy parecido, independientemente del servidor web o contenedor de Servlets que usemos. En caso de usar el *kit* de programación de Java de Sun, el JDK oficial, debemos asegurarnos de que nuestro `CLASSPATH`, la lista de librerías y directorios donde se buscarán las clases que usemos en nuestros programas contenga las librerías del API de Java Servlets. El nombre de esta librería cambia de versión en versión del API de Java, pero por lo general suele ser: `servlet-version.jar`. Una vez tengamos la librería de Servlets en nuestro `CLASSPATH`, el proceso de compilación de un Servlet es el siguiente:

```
javac ServletBasico.java
```

Debemos colocar el fichero `class` resultante en el directorio que nuestro contenedor de Servlets requiera para ejecutar dicho Servlet. Posteriormente, para probarlo, deberemos apuntar el navegador a la URL de nuestro Servlet, que estará formada, por un lado, por el directorio donde nuestro contenedor de Servlets muestra los Servlets (por ejemplo, `/servlets`) y, por otro, por el nombre del Servlet.

Por ejemplo, en JWS, el servidor de pruebas de Sun, los Servlets se colocan en el subdirectorio `servlets` del directorio de instalación del JWS y la URL se forma con:

```
http://servidor/servlet/ServletBasico
```

En Tomcat los Servlets se colocan en un directorio que indica la aplicación web que estemos desarrollando, en el subdirectorio `WEB-INF` y dentro de éste, en el subdirectorio `classes`. A continuación, si la aplicación web, por ejemplo, se llama `test` la URL resultante es:

```
http://servidor/test/servlets/ServletBasico
```

5.3.6. Generación de contenido desde los Servlets

Como hemos visto, el API nos proporciona una clase `PrintWriter` donde podemos enviar todos los resultados. No obstante, esto no es suficiente para que nuestro Servlet devuelva HTML al cliente.

ANOTACIONES

El primer paso para construir un Servlet que devuelva HTML al cliente es notificar al contenedor de Servlets que el retorno de nuestro Servlet es de tipo HTML. Hay que recordar que HTTP contempla la transferencia de múltiples tipos de datos mediante el envío de la etiqueta MIME de marcado de tipo: `Content-Type`. Para ello disponemos de un método que nos permite indicar el tipo retornado, `setContentType`. Por ello y antes de cualquier interacción con la respuesta, debemos marcar el tipo del contenido.

```
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;

public class Holaweb extends HttpServlet
{
 public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException
 {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n" +
      "<HTML>\n" +
      "<HEAD><TITLE>Hola</TITLE></HEAD>\n" +
      "<BODY>\n" +
      "<H1>Hola web</H1>\n" +
      "</BODY></HTML>");
 }
}
```

Como podemos ver, la generación del resultado en HTML constituye un trabajo realmente engorroso, especialmente si tenemos en cuenta que una parte de este HTML no varía de Servlet a Servlet o de ejecución en ejecución. La solución a este tipo de problemas pasa por utilizar JSP en lugar de Servlets, pero si por algún motivo debemos usar Servlets, existen algunas aproximaciones que nos pueden ahorrar trabajo. La principal solución consiste en declarar métodos que realmente devuelvan estas partes comunes del HTML: la línea `DOCTYPE`, la cabecera, e incluso, un encabezado y un pie común para todo el sitio web de la empresa.

Para ello, construiremos una clase que contenga una serie de utilidades que podremos usar en nuestro proyecto de aplicación web.

ANOTACIONES

```
public class Utilidades
{
 public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD"+
    " HTML 4.0 Transitional//EN\">";

 public static String cabeceraTítulo(String título) {
    return(DOCTYPE + "\n" +
       "<HTML>\n" +
       "<HEAD><TITLE>" + título + "</TITLE></HEAD>\n");
 }

 // Aquí añadiremos otras utilidades
}
```

5.3.7. Manejar datos de formularios

Obtener los datos que nos envía un usuario desde un formulario es una de las tareas más complejas y monótonas de la programación con CGI. Dado que tenemos dos métodos de paso de valores, `GET` y `POST`, cuyo funcionamiento es diferente, debemos desarrollar dos métodos para leer estos valores. Además, tenemos que analizar, *parsear* y decodificar las cadenas que contienen codificados valores y variables.

Una de las ventajas de usar Servlets es que el API de Servlets soluciona todos estos problemas. Esta tarea se realiza de forma automática y los valores se ponen a disposición del Servlet mediante el método `getParameter` de la clase `HttpServletRequest`. Este sistema de paso de parámetros es independiente del método usado por el formulario para pasar parámetros al Servlet (sea GET o POST). Disponemos, además, de otros métodos que nos pueden ayudar a recoger los parámetros enviados por el formulario. Por un lado, tenemos una versión de `getParameter` llamada `getParameters` que debemos usar en caso de que el parámetro buscado pueda tener más de un valor. Por otro lado, tenemos `getParameterNames`, que nos devuelve el nombre de los parámetros pasados.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
```

ANOTACIONES

```
public class DosParams extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException
  {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String tit= "Leyendo 2 Parametros";
     out.println(Utilidades.cabeceraTitulo(tit) +
        "<BODY>\n" +
        "<H1 ALIGN=CENTER>" + tit + "</H1>\n" +
        "<UL>\n" +
        " <LI>param1: "
        + request.getParameter("param1") + "\n" +
        " <LI>param2: "
        + request.getParameter("param2") + "\n" +
        "</UL>\n" +
        "</BODY></HTML>");
  }

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
       throws ServletException, IOException
  {
     doGet(request, response);
  }
}
```

Este ejemplo de Servlet lee dos parámetros llamados `param1`, `param2` y nos muestra sus valores en una lista de HTML. Podemos observar el uso de `getParameter` y cómo, haciendo que `doPost` llame a `doGet`, se consigue que la aplicación responda a los dos métodos. Disponemos, si fuera necesario, de métodos para leer la entrada estándar, al igual que en la programación de CGI.

Veremos ahora un ejemplo más complejo que ilustrará toda la potencia del API de Servlets. Este ejemplo recibe los datos de un formu-

ANOTACIONES

lario, busca los nombres de los parámetros y los pinta, indicando los que tienen valor cero y los de múltiples valores.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class Parametros extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws ServletException, IOException
  {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String tit = "Leyendo Parametros";

        out.println(Utilidades.cabeceraTitulo(tit) +
               "<BODY BGCOLOR=\"#FDF5E6\">\n" +
               "<H1 ALIGN=CENTER>" + tit + "</H1>\n" +
               "<TABLE BORDER=1 ALIGN=CENTER>\n" +
               "<TR BGCOLOR=\"#FFAD00\">\n" +
               "<TH>Nombre Parametro<TH>Valor Parametro(s)");

        // Leemos los nombres de los parametros
        Enumeration params = request.getParameterNames();

        // Recorremos el vector de nombres
        while(params.hasMoreElements())
        {
          // Leemos el nombre
          String param = (String)params.nextElement();

          // Imprimimos el nombre
          out.println("<TR><TD>" + paramName + "\n<TD>");

          // Leemos el vector de valores del parametro
          String[] valores = request.getParameterValues(param);

          if (valores.length == 1)
          {
             // Solo un valor o vacío
             String valor = valores[0];

             // Valor vacío.
             if (valor.length() == 0)
```

ANOTACIONES

```
                    out.print("<I>Vacío</I>");
            else
            out.print(valor);
          }
          else
          {
            // Múltiples valores
              out.println("<UL>");
           for(int i=0; i<valores.length; i++)
      {
           out.println("<LI>" + valores[i]);
      }
           out.println("</UL>");
      }
      }
        out.println("</TABLE>\n</BODY>\n</HTML>");
    }

    public void doPost(HttpServletRequest request,
                            HttpServletResponse response)
      throws ServletException, IOException
  {
      doGet(request, response);
  }
}
```

Primero buscamos los nombres de todos los parámetros mediante el método getParameterNames. Esto nos devuelve una Enumeration. A continuación, recorremos de forma estándar la Enumeration (usando hasMoreElements para determinar cuándo parar y usando nextElement para obtener cada entrada). Como nextElement devuelve un objeto de tipo Object, convertimos el resultado a String y los usamos con getParameterValues, obteniendo un vector de String. Si este vector sólo tiene una entrada y sólo contiene un String vacío, el parámetro no tiene valores, y el Servlet genera una entrada "Vacío" en cursiva. Si el vector es de más de una entrada, el parámetro tiene múltiples valores, que se muestran en una lista no ordenada. De otra forma, mostramos el único valor.

Tenemos aquí un formulario HTML que servirá para probar el Servlet, ya que le envía un grupo de parámetros. Como el formulario contiene

ANOTACIONES

un campo de tipo PASSWORD usaremos el método POST para enviar los valores.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML> <HEAD>
  <TITLE>Formulario con POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
  <H1 ALIGN="CENTER">Formulario con POST</H1>
  <FORM ACTION="/ejemplos/servlets/Parámetros" METHOD="POST">
     Código: <INPUT TYPE="TEXT" NAME="código"><BR>
     Cantidad: <INPUT TYPE="TEXT" NAME="cantidad"><BR>
     Precio: <INPUT TYPE="TEXT" NAME="precio" VALUE="\$"><BR>
     <HR>
     Nombre:
     <INPUT TYPE="TEXT" NAME="Nombre"><BR>
     Apellidos:
     <INPUT TYPE="TEXT" NAME="Apellidos"><BR>

     Dirección:
     <TEXTAREA NAME="dirección" ROWS=3 COLS=40></TEXTAREA><BR>

     Tarjeta de crédito:<BR>
        <INPUT TYPE="RADIO" NAME="tarj"
              VALUE="Visa">Visa<BR>
        <INPUT TYPE="RADIO" NAME="tarj"
              VALUE="Master Card">Master Card<BR>
        <INPUT TYPE="RADIO" NAME="tarj"
              VALUE="Amex">American Express<BR>
        <INPUT TYPE="RADIO" NAME="tarj"
              VALUE="Maestro">Maestro<BR>
  Número tarjeta:
  <INPUT TYPE="PASSWORD" NAME="numtar"><BR>

    Repetir Número tarjeta:
    <INPUT TYPE="PASSWORD" NAME="numtar"><BR><BR>
    <CENTER>
          <INPUT TYPE="SUBMIT" VALUE="Realizar Pedido">
    </CENTER>
  </FORM>
  </BODY>
  </HTML>
```

5.3.8. La solicitud HTTP: HttpRequest

Cuando un cliente HTTP (el navegador) envía una petición, puede enviar un cierto número de cabeceras opcionales, excepto Content-Length, que es requerida en las peticiones POST. Dichas ca-

ANOTACIONES

beceras proporcionan información adicional al servidor web, que puede usarla para adecuar más su respuesta a la petición del navegador.

Algunas de las cabeceras más comunes y útiles son:

- Accept. Los tipos MIME que prefiere el navegador.
- Accept-Charset. El conjunto de caracteres que acepta el navegador.
- Accept-Encoding. Los tipos de codificación de datos que acepta el navegador. Puede indicar, por ejemplo, que el navegador acepta las páginas comprimidas, etc.
- Accept-Language. El idioma que prefiere el navegador.
- Authorization. Información de autorización, usualmente en respuesta a una petición del servidor.
- Cookie. Los `cookies` almacenados en el navegador que correspondan al servidor.
- Host. Servidor y puerto de la petición original.
- If-Modified-Since. Sólo enviar si ha sido modificado desde la fecha indicada.
- Referer. La URL de la página que contiene el enlace que el usuario siguió para obtener la página actual.
- User-Agent. Tipo y marca del navegador, útil para adecuar la respuesta a navegadores específicos.

Para leer las cabeceras sólo tenemos que llamar al método `getHeader` de `HttpServletRequest`. Éste nos devolverá un `String` si en la petición se envió la cabecera indicada, y `null` si no se envió.

Existen algunos campos de la cabecera que son usados de forma tan habitual que tienen métodos propios. Disponemos del método

ANOTACIONES

getCookies para acceder a los *cookies* enviados con la petición HTTP, el cual los analiza y los almacena en un vector de objetos de tipo Cookie. Los métodos getAuthType y getRemoteUser nos permiten acceder a cada uno de los dos componentes del campo Authorization de la cabecera. Los métodos getDateHeader y getIntHeader leen la cabecera específica y la convierten a valores Date e int, respectivamente.

En vez de buscar una cabecera particular, podemos usar el getHeader-Names para obtener una Enumeration de todos los nombres de cabecera de esta petición particular. Si es así, es posible recorrer esta lista de cabeceras, etc.

Finalmente, además de acceder a los campos de la cabecera de la petición, podemos obtener información sobre la propia petición. El método getMethod devuelve el método usado para la petición (normalmente GET o POST, pero HTTP dispone de otros métodos menos habituales, como HEAD, PUT, y DELETE). El método getRequestURI devuelve la URI (la parte de la URL que aparece después del nombre del servidor y del puerto, pero antes de los datos del formulario). El getRequestProtocol nos devuelve el protocolo usado, generalmente "HTTP/1.0" o "HTTP/1.1".

5.3.9. Información adicional sobre la petición

Podemos, además de las cabeceras de la petición HTTP, obtener una serie de valores que nos proporcionarán información adicional sobre la petición. Algunos de estos valores estaban disponibles para la programación CGI como variables de entorno. Todos ellos se encuentran disponibles como métodos de la clase HttpRequest.

getAuthType (). Si se suministró una cabecera Authorization, éste es el esquema especificado (basic o digest). Variable CGI: AUTH_TYPE.

getContentLength (). Sólo para peticiones POST, el número de bytes enviados.

getContentType (). El tipo MIME de los datos adjuntos, si se específica. Variable CGI: CONTENT_TYPE.

getPathInfo (). Información del *Path* adjunto a la URL. Variable CGI: PATH_INFO.

getQueryString (). Para peticiones GET, son los datos enviados en forma de una cadena única, con los valores codificados. No es habitual su uso en los Servlets, al disponer de acceso directo a los parámetros ya decodificados. Variable CGI: QUERY_STRING.

getRemoteAddr (). La dirección IP del cliente. Variable CGI: REMOTE_ADDR.

getRemoteUser (). Si se suministró una cabecera Authorization, la parte del usuario. Variable CGI: REMOTE_USER.

getMethod (). El tipo de petición normalmente es GET o POST, pero puede ser HEAD, PUT, DELETE, OPTIONS, o TRACE. Variable CGI: REQUEST_METHOD.

5.3.10. Códigos de estado y respuesta

Al atender una petición web de un navegador, la respuesta suele contener un código numérico que indica al navegador el cumplimiento o no de su petición y las razones de que ésta no haya sido satisfecha. Además, incluye algunas cabeceras que proporcionan al navegador información adicional sobre la respuesta. Desde los Servlet podemos indicar el código de retorno de HTTP y el valor de algunas de estas cabeceras. Ello nos permite reenviar al usuario hacia otra página, indicar el tipo de contenido de la respuesta, solicitar un *password* al usuario, etc.

Códigos de estado

Para devolver un código de estado concreto, disponemos en nuestros Servlets del método `setStatus`. Dicho método indica al servidor web y al contenedor de Servlets el estado que deben devolver al cliente. El API de Servlets proporciona en la clase `HttpServletResponse` una tabla de constantes para facilitar

ANOTACIONES

el uso de los códigos de respuesta. Dichas constantes tienen nombres fáciles de usar y recordar.

Ejemplo

Por ejemplo, la constante para el código 404 (en el estándar HTTP calificado como *not found*, no encontrado), es SC_NOT_FOUND.

Si el código que devolvemos no es el de defecto (200, SC OK), debemos llamar a `setStatus` antes de usar el `PrintWriter` para devolver el contenido al cliente. Disponemos, además, de `setStatus` para devolver códigos de error de dos métodos más especializados: `sendError` para devolver errores (código 404), que permite añadir al código numérico un mensaje en HTML y `sendRedirect` (código 302), que permite especificar la dirección a la que se redirecciona al cliente.

Cabeceras de retorno

Además de incluir un código numérico al responder a la petición http, el servidor puede incluir una serie de valores en las cabeceras de las respuestas. Estas cabeceras permiten informar al navegador de la expiración de la información enviada (cabecera `Expires`), de que debe refrescar la información pasado un tiempo especificado (cabecera `Refresh`), etc. Desde nuestros Servlets podemos modificar el valor de estas cabeceras o añadir algunas. Para ello, disponemos del método `setHeader` de la clase `HttpServletResponse`, que nos permite asignar valores arbitrarios a las cabeceras que devolvemos al cliente. Igual que en el caso de los códigos de retorno, debemos seleccionar las cabeceras antes de enviar ningún valor al cliente. Disponemos de dos métodos auxiliares de `setHeader` para aquellos casos en los que deseemos enviar cabeceras que contengan fechas o enteros. Estos métodos, `setDateHeader` y `setIntHeader`, no evitan tener que convertir fechas y enteros a `String`, el parámetro aceptado por `setHeader`.

Disponemos, además, de unos métodos especializados para algunas cabeceras de uso más común:

ANOTACIONES

`SetContentType`. Proporciona un valor a la cabecera `Content-Type` y se debe usar en la mayoría de los Servlets.

`SetContentLength`. Permite asignar un valor a la cabecera `Content-Length`.

`AddCookie`. Asigna un *cookie* a la respuesta.

`SendRedirect`. Además de asignar el código de estado 302, como hemos visto, asigna la dirección a la que se redirige el usuario en la cabecera `Location`.

5.3.11. Seguimiento de sesiones

HTTP es un protocolo sin estado, lo cual significa que cada petición es totalmente independiente de la precedente, de manera que no podemos vincular dos peticiones consecutivas entre sí. Esto resulta catastrófico si deseamos usar la web para algo más que visualizar documentos. Si estamos desarrollando una aplicación de comercio electrónico como una tienda en línea, debemos tener un control sobre los productos que ha ido seleccionando nuestro cliente para que al llegar a la pantalla de pedido tengamos una lista de la compra correcta. Al llegar a esta pantalla, ¿cómo obtenemos la lista de los objetos seleccionados para comprar?

Existen tres aproximaciones a este problema:

1. Usar *cookies*. Las *cookies* son pequeños pedazos de información que el servidor envía al navegador y que éste le reenvía cada vez que accede al sitio web. A pesar de tener un magnífico soporte de *cookies*, utilizar esta técnica para realizar un seguimiento de la sesión sigue siendo una tarea engorrosa:

   - Controlar el *cookie* que contiene el identificador de sesión.
   - Controlar la expiración de este.
   - Asociar lo almacenado en el *cookie* con información de una sesión.

2. Reescritura de URL. Podemos utilizar la URL para añadir alguna información adicional que identifique la sesión. Esta solución pre-

ANOTACIONES

senta la ventaja de que funciona con navegadores en los que no hay soporte de *cookies* o en los que éste está desactivado. Sin embargo, sigue siendo un sistema engorroso:

- Debemos asegurarnos de que todas las URL que lleguen al usuario contengan la información de sesión adecuada.

- Dificulta que el usuario anote las direcciones en su libreta de direcciones (*bookmarks*), pues éstas contendrán información de sesión caducada.

3. Campos ocultos en los formularios. Podemos usar los campos de tipo `HIDDEN` de los formularios de HTML para propagar la información que nos interese. Evidentemente adolece de los mismos problemas que las anteriores soluciones.

Afortunadamente, el API de Servlets nos facilita una solución a este problema. Los Servlets proporcionan un API, `HttpSession`, de alto nivel que gestiona sesiones. La gestión la realiza mediante *cookies* y reescritura de URL. Esta API aísla al autor de los Servlets de los detalles de la gestión de sesiones.

Obtener la sesión asociada a la petición

Para obtener la sesión asociada a la petición HTTP en curso disponemos de un método `getSession` de la clase `HttpServletRequest`. Si existe una sesión, dicho método nos retorna un objeto de tipo `HttpSession`. En caso de no existir, nos devolverá `null`. Podemos llamar a `getSession` usando un parámetro adicional que, en caso de que la sesión no existiese, la crease automáticamente.

```
HttpSession sesion = request.getSession(true);
```

Acceder a la información asociada

Los objetos `HttpSession` que representan la información asociada a una sesión nos permiten almacenar en su interior una serie de valores con nombre. Para leer estos valores disponemos de `getAttribute`, mientras que para modificarlos contamos con `setAttribute`.

ANOTACIONES

Nota

En versiones anteriores a la 2.2 del API de Servlets, las funciones de acceso a la información de sesión eran: `getValue` y `setValue`.

Un esquema de acceso a estos datos de sesión sería:

```
HttpSession sesion = request.getSession(true);
 String idioma= (String)session.getAttribute("idioma");
 if (idioma == null)
 {
    idioma=new String("español");
    response.setAttribute("idioma",idioma);
 }
 // a partir de aquí mostraríamos los datos en el idioma
 // preferido por el usuario
```

Disponemos de métodos para acceder a la lista de atributos guardados en la sesión; a saber, `getAttributeNames`, que devuelve una `Enumeration`, de forma similar a los métodos `getHeaders` y `getParameterNames`, de `HttpServletRequest`.

Nota

En versiones anteriores a la 2.2 del API de Servlets, la función de listado de nombres de valores era `getValueNames`.

Disponemos, además, de algunas funciones de utilidad para acceder a la información de las sesiones:

getId devuelve un identificador único generado para cada sesión.

isNew devuelve *cierto* si el cliente nunca ha visto la sesión porque acaba de ser creada.

getCreationTime devuelve la hora, en milisegundos desde 1970, año en que se creó la sesión.

getLastAccessedTime devuelve la hora, en milisegundos desde 1970, año en que la sesión fue enviada por última vez al cliente.

5.3.12. Java Server Pages: JSP

Las Java Server Pages, en adelante JSP, son una extensión de HTML desarrollada por Sun que permite incrustar en el código HTML instrucciones (*scriptlets*) de Java. Esto nos permite una mayor simplicidad a la hora de diseñar sitios web dinámicos. Podemos usar cualquiera de los múltiples editores de HTML para crear nuestra web, o bien podemos dejar esta tarea en manos de un diseñador y cen-

tramos en el desarrollo del código Java que generará las partes dinámicas de la página, para posteriormente incrustar este código en la página.

Un ejemplo simple de página JSP que nos introducirá algunos de los conceptos básicos del estándar es la siguiente:

```
<HTML>
 <BODY>
 <H1>Bienvenido. Día: < %= fecha %> </H1>
 <B>
 < % if(nombre==null)
       out.println("Nuevo Usuario");
    else
       out.println("Bienvenido de nuevo");
 %>
 </B>
 </BODY>
</HTML>
```

Las páginas JSP normalmente tienen extensión `.jsp` y suelen colocarse en el mismo directorio que los ficheros HTML. Como podemos ver, una página `.jsp` no es más que una página HTML en la que incrustamos trozos de código Java, delimitados por `< %` y `%>`. Las construcciones delimitadas por `< %` y `%>` pueden ser de tres tipos:

- Elementos de *script* que nos permiten introducir un código que formará parte del Servlet que resulte de la traducción de la página.
- Las directivas nos permiten indicar al contenedor de Servlets cómo queremos que se genere el Servlet.
- Las acciones nos permiten especificar componentes que deberán ser usados.

Cuando el servidor/contenedor de Servlets procesa una página JSP, convierte ésta en un Servlet en el que todo el HTML que hemos introducido en la página JSP es impreso en la salida, para a continuación compilar este Servlet y pasarle la petición. Este paso de conversión/

ANOTACIONES

compilación generalmente se realiza solamente la primera vez que se accede a la página o si el fichero JSP ha sido modificado desde la última vez que se compiló.

Elementos de *script*

Los elementos de *script* nos permiten insertar código Java dentro del Servlet resultante de la compilación de nuestra página JSP. Tenemos tres opciones de inserción de código:

– Expresiones de la forma `< %= expresion %>` que son evaluadas e insertadas en la salida.

– *Scriptlets* de la forma `< % codigo %>` que se insertan dentro del método Service del Servlet.

– Declaraciones de la forma `< %! codigo %>` que se insertan en el cuerpo de la clase del Servlet, fuera de cualquier método existente.

• Expresiones

Las expresiones JSP se utilizan para insertar en la salida un valor de Java directamente. Su sintaxis es:

```
< %= expresion %>
```

La expresión se evalúa, obteniéndose un resultado que se convierte en una cadena que se insertará en la página resultante. La evaluación se realiza en tiempo de ejecución, al solicitarse la página. Por ello las expresiones pueden acceder a los datos de la petición HTTP. Por ejemplo,

```
< %= request.getRemoteUser() %> entró el
< %= new java.util.Date() %>
```

Este código mostrará el usuario remoto (si está autenticado) y la fecha en que se solicitó la página.

Podemos ver en el ejemplo que estamos utilizando una variable, `request`, que representa la petición de HTTP. Esta variable prede-

finida pertenece a un conjunto de variables predefinidas de las que podemos servirnos:

– `request`: el `HttpServletRequest`
– `response`: el `HttpServletResponse`
– `session`: el `HttpSession` asociado con el `request` (si existe)
– `out`: el `PrintWriter` usado para enviar la salida al cliente

Existe una sintaxis alternativa para introducir expresiones. Dicha sintaxis se introdujo para compatibilizar JSP con editores, *parsers*, etc., de XML. Está basado en el concepto de *tagActions*. La sintaxis para una expresión es:

```
<jsp:expression> expresión</jsp:expression>
```

• *Scriptlets*

Los *scriptlets* nos permiten insertar código Java arbitrario en el Servlet que resultará de la compilación de la página JSP. Un *scriptlet* tiene la siguiente forma:

```
< % codigo %>
```

En un *scriptlet* podemos acceder a las mismas variables predefinidas que en una expresión. Por ejemplo:

```
< %
 String user = request.getRemoteUser();
 out.println("Usuario: " + user);
%>
```

Los *scriptlets* se insertan en el Servlet resultante de forma literal respecto a cómo están escritos, mientras que el código HTML introducido se convierte en `println`. Ello nos permite realizar construcciones como la siguiente:

```
<% if (obtenerTemperatura() < 20) { %>
      <B>¡Abrígate! ¡Hace frío! </B>
<% } else { %>
      <B>¡Que tengas buen día!</B>
< % } %>
```

ANOTACIONES

Observaremos en el ejemplo que los bloques de código Java pueden afectar e incluir el HTML definido en las páginas JSP. El código anterior, una vez compilada la página y generado el Servlet, se convertirá en algo similar a esto:

```
if (obtenerTemperatura() < 20) {
     out.println("<B>¡Abrígate! ¡Hace frío! </B>");
} else {
     out.println("<B>¡Que tengas buen día!</B>");
}
```

El equivalente en XML para los *scriptlets* es:

```
<jsp:scriptlet> código </jsp:scriptlet>
```

- **Declaraciones JSP**

Las declaraciones nos permiten definir métodos o campos que serán posteriormente insertados en el Servlet fuera del método `service`. Su forma es como:

```
< %! codigo %>
```

Las declaraciones no generan salida. Por ello suelen usarse para definir variables globales, etc. Por ejemplo, el siguiente código incluye un contador en nuestra página:

```
< %! private int visitas = 1; %>
Visitas a la página durante el funcionamiento del servidor:
< %= visitas++ %>
```

Indicar que este contador se pone a uno cada vez que el contenedor de Servlets se reinicia o cada vez que modificamos el Servlet o el fichero JSP (lo que obliga al servidor a recargarlo). El equivalente a las declaraciones para XML es:

```
<jsp:declaration> código </jsp:declaration>
```

ANOTACIONES

Directivas JSP

Las directivas afectan a la estructura general de la clase Servlet. Su forma es la siguiente:

```
< %@ directiva atributo1="valor1"
                 atributo2="valor2"
                 ...
                 %>
```

Disponemos de tres directivas principales:

`page` Que nos permite modificar la compilación de la página JSP a Servlet.

`include` Que nos permite insertar en el Servlet resultante otro fichero (esta inserción se realiza en el momento de la traducción de JSP a Servlet).

`taglib` Que nos permite indicar qué librerías de etiquetas deseamos usar. JSP nos permite definir nuestras propias librerías de etiquetas.

Ejemplo

Por ejemplo, podemos importar clases, modificar la clase del Servlet, etc.

- **La directiva page**

Podemos definir usando page los siguientes atributos que modificarán la traducción del JSP a Servlet:

– `import="paquete.clase"` o bien `import="paquete.clase1, ...,paquete.claseN"`. `Import` nos permite especificar los paquetes y clases que deberían ser importados por Java al compilar el Servlet resultante. Este atributo puede aparecer múltiples veces en cada JSP. Por ejemplo:

```
< %@ page import="java.util.*" %>
< %@ page import="edu.uoc.campus.*" %>
```

– `contentType="MIME-Type"` o bien `contentType="MIME-Type; charset=Character-Set"` Esta directiva nos permite

ANOTACIONES

especificar el tipo MIME resultante de la página. El valor por defecto es `text/html`. Por ejemplo:

```
< %@ page contentType="text/plain" %>
```

Esto es equivalente a usar el *scriptlet*:

```
< % response.setContentType("text/plain"); %>
```

– `isThreadSafe="true|false"`. Un valor de `true` (el de defecto) indica que el Servlet resultante será un Servlet normal, donde múltiples peticiones pueden procesarse simultáneamente, suponiendo que las variables de instancia compartidas entre *threads* serán sincronizadas por el autor. Un valor de `false` indica que el Servlet debería implementar un modelo de `SingleThreadModel`.

– `session="true|false"`. Un valor de `true` (por defecto) indica que debería existir una variable predefinida `session` (de tipo `HttpSession`) con la sesión o en caso de no existir sesión, debería crearse una. Un valor de `false` indica que no se usarán sesiones y los intentos de acceder a ellas resultarán en errores en el momento de traducción a Servlet.

– `extends="package.class"`. Esto indica que el Servlet generado deberá extender una superclase diferente. Debemos usarla con extrema precaución, ya que el contenedor de Servlets que usemos podría requerir el uso de una superclase concreta.

– `errorPage="URL"`. Especifica qué página JSP se procesará si se lanza cualquier excepción (un objeto de tipo `Throwable`) y no se captura en la página actual.

– `isErrorPage="true|false"`. Indica si la página actual es o no una página de tratamiento de errores.

La sintaxis XML equivalente es:

```
<jsp:directive.Directiva atributo=valor />
```

ANOTACIONES

Por ejemplo, las dos líneas siguientes son equivalentes:

```
< %@ page import="java.util.*" %>
<jsp:directive.page import="java.util.*" />
```

- **La directiva `include`**

La directiva `include` nos permite incluir ficheros en la página JSP en el momento en que ésta es traducida a Servlet. La sintaxis es la siguiente:

```
< %@ include file="fichero que se vaya a incluir" %>
```

El fichero que se vaya a incluir puede ser relativo a la posición del JSP en el servidor, por ejemplo, `ejemplos/ejemplo1.jsp` o absoluto, por ejemplo, `/general/cabec.jsp`. Y puede contener cualquier construcción de JSP: html, scriptlets, directivas, acciones, etc.

La directiva `include` puede suponer un gran ahorro de trabajo, pues nos puede permitir escribir en un sólo fichero cosas como los menús de las páginas de nuestro sitio web, de manera que sólo tengamos que incluirlos en cada JSP que utilicemos.

1.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 <HTML>
   <HEAD>
      <TITLE>Sitio web</TITLE>
      <META NAME="author" CONTENT="carlesm@asic.udl.es">
      <META NAME="keywords" CONTENT="JSP, Servlets">
      <META NAME="description" CONTENT="Una página">
      <LINK REL=STYLESHEET HREF="estilo.css" TYPE="text/css">
   </HEAD>
   <BODY>
```

2.

```
<HR>
 <CENTER><SMALL>&copy; Desarrollador del web, 2003. Todos los
 derechos reservados</SMALL></CENTER>
</BODY> </HTML>
```

3.

```
< %@ include file="/cabecera.html" %>
 <!-- Página JSP -->
    .
    .
< %@ include file="/pie.html" %>
```

ANOTACIONES

En este ejemplo tenemos tres ficheros, respectivamente: `cabecera.html`, `pie.html` y una página JSP del sitio web. Como podemos ver, tener en ficheros aparte un fragmento del contenido de la página simplifica sobremanera el mantenimiento y la escritura de páginas JSP.

Un punto que debemos subrayar es que la inclusión se realiza en el momento en que se traduce la página JSP a Servlet. Si cambiamos alguna cosa en los ficheros incluidos, debemos forzar la retraducción de todo el sitio web. Esto, pese a parecer un problema, se ve ampliamente compensado por el beneficio que supone en cuanto a eficiencia la obligación de incluir los ficheros sólo una vez.

Disponemos, en caso de que queramos que se incluyan de nuevo en cada petición, de una alternativa que nos viene dada por la versión XML de la directiva:

```
<jsp:include file="/cabecera.html">
 <!-- Página JSP -->
   .
   .
<jsp:include file="/cabecera.html">
```

En este caso, la inclusión se realiza en el momento de servir la página, pero en este caso no podemos incluir ningún tipo de JSP en el fichero que vayamos a incluir, debiendo ser éste tan sólo HTML.

Variables predefinidas

Disponemos en los JSP de un grupo de variables ya definidas que nos facilitan el desarrollo de código.

- `request`. El objeto `HttpServletRequest` asociado a la petición. Nos permite acceder a los parámetros de la petición (mediante `getParameter`), el tipo de petición y las cabeceras HTTP (*cookies*, `Referer`, etc.).
- `response`. Éste es el objeto `HttpServletResponse` asociado a la respuesta del Servlet. Dado el objeto de *stream* de salida

ANOTACIONES

(la variable `out` definida más adelante) tiene un *buffer*, podemos seleccionar los códigos de estado y las cabeceras de respuesta.

- `out`. Éste es el objeto `PrintWriter` usado para enviar la salida al cliente.

- `session`. Éste es el objeto `HttpSession` asociado con la petición. Las sesiones se crean automáticamente por defecto. Esta variable existe incluso si no hubiera una sesión de referencia. La única excepción es si usamos el atributo `session` de la directiva `page`.

- `application`. Éste es el objeto `ServletContext` obtenido mediante `getServletConfig().getContext()`.

- `config`. El objeto `ServletConfig` para esta página.

Acciones

Las acciones JSP utilizan construcciones con sintaxis válida de XML para controlar el comportamiento del contenedor de Servlets. Las acciones permiten insertar ficheros dinámicamente, utilizar componentes JavaBeans, reenviar al usuario a otra página, etc.

- **`jsp:include`**

Esta acción nos permite insertar ficheros en la página que está siendo generada. La sintaxis es:

```
<jsp:include page="relative URL" flush="true" />
```

A diferencia de la directiva `include`, que inserta el fichero en el momento de la conversión de la página JSP a Servlet, esta acción inserta el fichero en el momento en que la página es solicitada. Por un lado, esto supone una menor eficiencia e impide a la página incluida contener un código JSP, pero por otro lado, esto supone un aumento de la flexibilidad, ya que podemos cambiar los ficheros insertados sin necesidad de recompilar las páginas.

ANOTACIONES

Ejemplo

Aquí tenemos un ejemplo de página que inserta un fichero de noticias en la página web. Cada vez que queremos cambiar las noticias, sólo tenemos que cambiar el fichero incluido, cosa que podemos dejar en manos de los redactores sin necesidad de recompilar los ficheros JSP.

```
< %@ include file="/cabecera.html" %>
Ultimas noticias:
<jsp:include page="news/noticias.html" />
< %@ include file="/pie.html" %>
```

- **jsp:useBean**

Esta acción nos permite cargar un JavaBean en la página JSP para utilizarlo. Se trata de una capacidad muy útil, porque nos permite utilizar la reusabilidad de las clases Java. La forma más simple para especificar que se debería usar un Bean es:

```
<jsp:useBean id="nombre" class="paquete.clase" />
```

El significado de este código es: instancia un objeto de la clase especificada por `clase`, y asignarlo a la variable llamada `id`. Podemos también añadir un atributo `scope` que indica que dicho `Bean` debe asociarse a más de una página.

Una vez tenemos el `Bean` instanciado, podemos acceder a sus propiedades. Es posible hacerlo desde de un *scriptlet* o bien usar dos acciones: `jsp:setProperty` y `jsp:getProperty`.

Describiremos las acciones mencionadas, `jsp:setProperty` y `jsp:getProperty`, con detalle más adelante. Por ahora nos bastará con saber que disponen de un atributo `param` para especificar qué propiedad queremos.

Tenemos aquí un pequeño ejemplo de uso de Beans desde páginas JSP:

Nota

Recordad los JavaBeans. Una propiedad **X** de tipo **Y** de un `Bean` implica: un método `getX ()` que devuelve un objeto de tipo **Y** y un método `setX (Y)`.

ANOTACIONES

Nota

Ubicación de los Bean.

Para que la carga de los Bean sea correcta, debemos asegurarnos de que el contenedor de Servlets los encontrará. Para ello, debemos consultar la documentación sobre cuál es la ubicación donde debemos colocarlos.

```
< %@ include file="/cabecera.html" %>
Ultimas noticias:

 <jsp:useBean id="mens" class="BeanMensaje" />
 <jsp:setProperty name="mens"
                   property="texto"
                   value="Hola" />
      <H1>Mensaje: <I>
      <jsp:getProperty name="mens" property="texto" />
      </I></H1>

< %@ include file="/pie.html" %>
```

El código del Bean utilizado para el ejemplo es el siguiente:

```
public class BeanMensaje
{
 private String texto = "No hay texto";

 public String getTexto()
 {
      return(texto);
 }

 public void setTexto(String texto)
 {
      this.texto = texto;
 }
}
```

La acción `jsp:useBean` nos proporciona además otras facilidades para trabajar con Beans. Si deseamos ejecutar un código específico en el momento en que el Bean es instanciado (es decir, es cargado por primera vez), podemos utilizar la siguiente construcción:

```
<jsp:useBean ...>
 código
</jsp:useBean>
```

Conviene recordar que los `Bean` pueden ser compartidos entre diferentes páginas. Sin embargo, no todos los usos de `jsp:useBean` resultan en la instanciación de un objeto nuevo. Por ejemplo:

```
<jsp:useBean id="mens" class="BeanMensaje" >
 <jsp:setProperty name="mens"
                   property="texto"
                   value="Hola" />
</jsp:useBean>
```

ANOTACIONES

Además de los explicados, useBean dispone de otros atributos menos empleados. Una referencia de todos ellos es:

- id. Da un nombre a la variable a la que asignaremos el Bean. Instanciará un nuevo objeto en caso de no encontrar uno con el mismo id y scope. En ese caso se utilizará el ya existente.

- class. Designa el nombre completo del paquete del Bean.

- scope. Indica el contexto en el que el Bean estará disponible. Disponemos de cuatro posibles ámbitos de disponibilidad:

  – page: indica que el Bean estará sólo disponible para la página actual. Esto significa que se almacenará en el PageContext de la página actual.

  – request: el Bean sólo está disponible para la petición actual del cliente, almacenado en el objeto ServletRequest.

  – session: nos indica que el objeto está disponible para todas las páginas durante el tiempo de vida de la HttpSession actual.

  – application: indica que está disponible para todas las páginas que compartan el mismo ServletContext.

  La importancia del ámbito estriba en que una entrada jsp:useBean sólo resultará en una instanciación de un nuevo objeto si no existían objetos anteriores con el mismo id y scope.

- type. Especifica el tipo de la variable a la que se referirá el objeto. Éste debe corresponder con el nombre de la clase o bien ser una superclase o una interfase que implemente la clase. Recordad que el nombre de la variable se designa mediante el atributo id.

- beanName. Da el nombre del Bean, como lo suministraríamos en el método *instantiate* de Beans. Está permitido suministrar un type y un BeanName, y omitir el atributo class.

ANOTACIONES

• **jsp:getProperty**

Esta acción recoge el valor de una propiedad del Bean, lo convierte en una cadena, e inserta dicho valor en la salida. Dispone de dos atributos requeridos, que son:

– name: el nombre de un Bean cargado anteriormente mediante jsp:useBean.

– property: la propiedad del Bean cuyo valor deseamos obtener.

El siguiente código muestra el funcionamiento de jsp:getProperty.

```
<jsp:useBean id="bean" ... />
<UL>
 <LI>Cantidad:
    <jsp:getProperty name="bean" property="cantidad" />
 <LI>Precio:
    <jsp:getProperty name="bean" property="precio" />
</UL>
```

• **jsp:setProperty**

La acción jsp:setProperty nos permite asignar valores a propiedades de Beans ya cargados. Tenemos dos opciones para asignar estos valores:

– En el momento de la instanciación. Podemos usar jsp:setProperty en el momento en que instanciemos un Bean, de tal manera que esta asignación de valores sólo se producirá una vez durante la vida del Bean:

```
<jsp:useBean id="mens" class="BeanMensaje" >
 <jsp:setProperty name="mens"
                     property="texto"
                     value="Hola" />
</jsp:useBean>
```

– En cualquier punto de nuestro código. Si usamos jsp:setProperty fuera de un contexto de jsp:useBean el valor se asignará a la propiedad, independientemente de si es la primera instanciación o si el Bean ya había sido instanciado anteriormente.

ANOTACIONES

```
<jsp:useBean id="mens" class="BeanMensaje" />
.....
<jsp:setProperty name="mens"
                   property="texto"
                   value="Hola" />
....
<jsp:setProperty name="mens"
                   property="texto"
                   value="Adiós" />
```

La acción `jsp:setProperty` dispone de cuatro posibles atributos:

- `name`. Este atributo designa el Bean cuya propiedad va a ser modificada.
- `property`. Este atributo indica la propiedad sobre la que queremos operar. Hay un caso especial: un valor de `"*"` significa que todos los parámetros de la petición de HTTP cuyos nombres correspondan con nombres de propiedades del Bean serán pasados a los métodos de selección apropiados.
- `value`. Este atributo opcional especifica el valor para la propiedad. Los valores son convertidos automáticamente a mediante el método estándar `valueOf` en la fuente o la clase envolvente. No podemos usar `value` y `param` juntos, pero podemos omitir ambas.
- `param`. Este parámetro opcional designa que un parámetro de la petición HTTP servirá para dar valor a la propiedad. En caso de que la petición HTTP no tenga dicho parámetro, el sistema no llama al método `setX` de propiedad del Bean.

El siguiente código pasa, si existe, el valor del parámetro `numItems` al Bean, para que éste lo asigne a su propiedad `numeroItems`.

```
<jsp:setProperty name="pedido"
                   property="numeroItems"
                   param="numItems" />
```

Si en la acción `jsp:setProperty` omitimos `value` y `param`, el contenedor asignará a la propiedad especificada el valor del parámetro de la petición HTTP que tenga un nombre idéntico. Usando la capacidad de no especificar la propiedad que se vaya a

ANOTACIONES

asignar (mediante el uso de "*"), podemos asignar fácilmente a un Bean todas las propiedades que correspondan a parámetros de la petición HTTP.

```
<jsp:setProperty name="pedido"
                    property="*" />
```

- **jsp:forward**

Esta acción nos permite reenviar la petición realizada a otra página. Dispone de un sólo parámetro `page` que contendrá la URL de destino. Podemos usar valores estáticos o bien usar un valor generado dinámicamente.

```
<jsp:forward page="/enconstruccion.jsp" />
<jsp:forward page="< %= urlDestino %>" />
```

- **jsp:plugin**

Esta acción permite insertar un elemento OBJECT o EMBED específico del navegador para especificar que el navegador debería ejecutar un applet usando el Plug-in Java.

5.4. Otras opciones de contenido dinámico

Además de las tecnologías vistas hasta el momento, disponemos de otros sistemas, tecnologías y lenguajes concebidos para desarrollar contenido web dinámico.

Uno de los sistemas más utilizados, como alternativa a los vistos es `mod_perl`, un módulo del servidor Apache que permite escribir páginas web utilizando el lenguaje de programación Perl de una forma similar a como se usa PHP. Este módulo presenta una serie de ventajas evidentes respecto a la escritura de CGI en Perl:

- Mejor uso de memoria. Su comportamiento es muy parecido al de PHP, ya que el módulo de Perl se inicia una sola vez, al arrancar

ANOTACIONES

el servidor web, permaneciendo en memoria a partir de ese momento. Ello evita el problema que suponía el hecho de tener que iniciar Perl para cada CGI.

- Mayor velocidad de respuesta. El hecho de tener el módulo precargado da una mayor agilidad a la respuesta, como lo hacen los programas de Perl ya precompilados (Perl precompila el código a un código intermedio que posteriormente interpreta).

- Permite a los programas un acceso más directo a información del servidor. El módulo proporciona una pasarela más eficiente y rica que la facilitada por las variables de entorno de CGI.

- Permite escribir extensiones del servidor completamente en Perl.

Dos de las grandes ventajas de `mod_perl` son el fuerte incremento de rendimiento y velocidad de los programas, por un lado, y el hecho de que para convertir un programa CGI escrito en Perl en uno para `mod_perl` sólo hacen falta unos retoques mínimos. Esas dos ventajas lo convierten en una opción muy válida para aquellas situaciones en las que ya tenemos un cierto número de programas CGI escritos en Perl. Una de las desventajas de `mod_perl` reside en el hecho de que solo está disponible para servidores Apache, con lo que si en nuestro entorno de trabajo no existe la posibilidad de usar Apache, no representará una alternativa válida.

Muchos servidores web, inclusive algunos de los vistos, como Roxen, proporcionan mecanismos de programación con una filosofía similar a la de JSP. Concretamente, Roxen, quizás uno de los de mayor riqueza de opciones para desarrollar aplicaciones, nos ofrece la posibilidad de extender nuestras páginas HTML con:

- Código RXML, una extensión de HTML de Roxen que incorpora todos los elementos de un lenguaje de programación: condicionales, bucles, etc., además de una rica librería de funciones que incluyen cosas como: acceso a bases de datos, a directorios LDAP, comunicaciones, gráficos, manejo de cadenas, etc.

- Código escrito en Pike, el lenguaje orientado a objetos con el que está desarrollado Roxen.

ANOTACIONES

- Código PHP, igualando en este caso las facilidades de Apache.

- Código Perl, sin llegar a ofrecer las mismas prestaciones que ofrece `mod_perl`, pero proporcionando un buen nivel de opciones.

Al igual que Roxen, tanto AOLServer como Apache nos permiten desarrollar módulos de extensión del servidor que permitirían tratar nuevas etiquetas de HTML, nuevas peticiones o protocolos de comunicaciones, etc. Algunos de estos sistemas permiten desarrollar extensiones de programación como lenguajes de plantilla, con una filosofía similar a JSP. Es el caso de Mason, DTL, etc.

Otras opciones pasan por usar un servidor "complejo" que incluya en un sólo producto mecanismos de extensión (con un lenguaje propio o uno de uso general), mecanismos de desarrollo de contenido dinámico y de páginas dinámicas. Uno de los más conocidos es Zope, el cual, basado en el lenguaje de programación Python, es un servidor de aplicaciones de código libre para construir portales, aplicaciones web, gestores de contenido, etc. Proporciona al programador gran cantidad de facilidades de desarrollo con un API rico y potente para proceso de peticiones HTTP, acceso a bases de datos, etc.

Finalmente, cabe destacar la existencia de opciones de más alto nivel, muchas de ellas basadas en alguno de los productos anteriores y diseñadas para el desarrollo de aplicaciones web complejas. Algunas de ellas, como Hendirá (http://www.enhydra.org) están basadas en JSP/Servlets (Enhydra es además un excelente contenedor de Servlets). Otras, como OpenACS, son aplicaciones muy orientadas a un tipo de sitios web concretos.

Ejemplo

OpenACS, por ejemplo, está concebida para desarrollar sitios web de tipo comunidad, portal, etc.

Uno de los puntos débiles de OpenACS es su dependencia de AOLServer y TCL. Ésta es una característica común de los paquetes de muy alto nivel, que suelen estar muy vinculados a algún servidor web concreto. Por otro lado, OpenACS ofrece una gran variedad de módulos y facilidades para crear sitios web.

Disponemos, además, de una rica variedad de productos del tipo CMS (*Content Management Systems*) para la mayoría de servidores web de código libre y contenedores de Servlets que proporcionan funcionalidades de modificación, adaptación y programación sufi-

cientes para algunos de los proyectos más complejos. En estos casos no se requiere la complejidad asociada a un desarrollo completo de una aplicación web.

5.5. Prácticas: creación de una aplicación simple con las técnicas presentadas

Vamos a crear un formulario que recoja saludos y los presente en pantalla. Para ello usaremos dos de las técnicas presentadas: CGI y Servlets.

5.5.1. CGI

Para escribir nuestro programa CGI hemos escogido el lenguaje de programación Perl. El código del programa es:

```
#!/usr/bin/perl
print "Content-type: text/html\n\n\n";
print "<html>\n";
print "<body>\n";
$QS=$ENV{"QUERY_STRING"};
if ($QS ne "")
{
 @params=split /&/,$QS;
 foreach $param (@params)
 {
     ($nom,$val)=split /=/,$param;
     $val=~s/\+/ /g;
     $val =~ s/ %([0-9a-fA-F]{2})/chr(hex($1))/ge;
     if($nom eq "saludo")
     {
           open FIT,">>lista";
           print FIT "$val\n";
           close FIT;
     }
 }
}
 open FIT,"<lista";
 while(<FIT>)
 {
     print "$_\n<HR>\n";
```

ANOTACIONES

```
 }
 close FIT;

print << "EOF";
<FORM METHOD=GET ACTION="visita.cgi">
Saludo: <INPUT TYPE=Text NAME="saludo" SIZE=40>
<BR>
<INPUT TYPE="submit" NAME="ENVIAR" VALUE="ENVIAR">
</FORM>
</BODY>
</HTML>
EOF
```

Como podemos ver, en este caso debemos procesar la variable de entorno que contiene los parámetros de forma manual.

5.5.2. Servlet Java

Veremos ahora una implementación equivalente usando Servlets de Java:

```
import java.io.*; import java.text.*; import java.util.*; import
javax.servlet.*; import javax.servlet.http.*;
/**
 * Procesamiento de un Formulario
 *
 * @author Carles Mateu
 */
 public class Formulario extends HttpServlet {

     Vector saludos;

     public void init(ServletConfig sc)
             throws ServletException
     {

          saludos=new Vector();

     }

     void listarSaludos(PrintWriter out)
     {
     for (Enumeration e = saludos.elements() ; e.hasMoreElements() ;)
    {
          out.println(e.nextElement()+"<HR>");
          }
     }

    public void doGet(HttpServletRequest request,
```

ANOTACIONES

```
                        HttpServletResponse response)
            throws IOException, ServletException
        {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
        String saludo=request.getParameter("saludo");

        if(¡saludo!=null)
        {
            saludos.add(saludo);
        }
            out.println("<html>");
            out.println("<body bgcolor=\"white\">");
        listarSaludos(out);
        out.println("<FORM METHOD=GET ACTION=\"Formulario\">\n"+
              "Saludo: <INPUT TYPE=Text NAME=\"saludo\" SIZE=40>\n"+
              "<BR> <INPUT TYPE=\"submit\" "+




            "NAME=\"ENVIAR\" VALUE=\"ENVIAR\">\n"+
            "</FORM> </BODY> </HTML>\n");
       out.println("</body>");
       out.println("</html>");
   }
}
```

En este caso, la persistencia de los datos es sólo la del Servlet, pues los datos se guardan en memoria.

ANOTACIONES

6. Acceso a bases de datos: JDBC

Una de las necesidades más perentorias con las que nos enfrentamos al desarrollar aplicaciones para Internet con Java, y para muchas otras aplicaciones, es la de disponer de un almacén de datos potente, robusto, rápido y fácilmente accesible. Para ello existen multitud de sistemas gestores de bases de datos (SGBD) de código libre. Analizaremos algunos de ellos en las siguientes secciones y presentaremos posteriormente una tecnología que proporciona Java para acceder a SGBD, especialmente relacionales, llamada JDBC (algunas veces erróneamente expandido como *Java database connectivity*).

6.1. Introducción a las bases de datos

Uno de los puntos críticos en el desarrollo de aplicaciones web es la elección del SGBD que utilizaremos. Existen en la actualidad bastantes SGBD de código libre, muchos de ellos del mismo nivel cualitativo que algunos de los SGBD comerciales más conocidos.

La mayoría de los SGBD de código libre proceden de dos fuentes principales: por un lado, de proyectos que nacieron como código libre desde su principio (proyectos de investigación, etc.) y por el otro, tenemos SGBD que pertenecían a empresas de software propietario, cuyo negocio principal no es el SGBD. Estas compañías optan por poner el producto bajo el amparo de una licencia de código libre, abriendo así su desarrollo a la comunidad. Veremos a continuación algunos de los SGBD de código libre más emblemáticos.

6.1.1. PostgreSQL

PostgreSQL (o Postgres) es uno de los SGBD más veteranos y conocidos del mundo del código libre. Arranca a mediados de los ochenta en la Universidad de Berkeley, bajo el nombre Postgres, a raíz de

ANOTACIONES

la investigación del grupo de bases de datos de Berkeley (especialmente de Michael Stonebraker). Postgres fue evolucionando hasta llegar a Postgres 4.2 en 1994. Postgres no usaba SQL como lenguaje de consulta, sino Postquel, un lenguaje propio. En 1995 Andrew Yu y Jolly Chen añadieron un intérprete SQL a Postgres 4.2, dando lugar al nacimiento de Postgres95, un producto ya bajo licencia de código libre y que salió de Berkeley para convertirse en un desarrollo llevado a través de todo Internet. En 1996 se eligió un nuevo nombre que resistiera el paso de los años y reflejara la relación del proyecto con el original Postgres (aún disponible) y las nuevas diferencias (básicamente, el uso de SQL). Nace así PostgreSQL.

A partir de ese momento, PostgreSQL se ha convertido en una de las bases de datos de elección en infinidad de proyectos, proporcionando al usuario algunas prestaciones del mismo nivel que las que ofrecen sistemas gestores de bases de datos comerciales como Informix y Oracle.

Las características más destacadas de PostgreSQL son:

- Soporte para transacciones.
- Subconsultas.
- Soporte de vistas.
- Integridad referencial.
- Herencia de tablas.
- Tipos definidos por el usuario.
- Columnas como vectores que pueden almacenar más de un valor.
- Añadir campos a tablas en tiempo de ejecución.
- Funciones de agregación (como sum() o count()) definibles por el usuario.

ANOTACIONES

- *Triggers*, comandos SQL que deben ejecutarse al actuar sobre una tabla.

- Tablas de sistema que podemos consultar para obtener información de tablas, de la base de datos y del motor de bases de datos.

- Soporte de objetos binarios grandes (mayores de 64 KB).

6.1.2. MySQL

MySQL se disputa con PostgreSQL el puesto de SGBD más conocido y usado de código libre. MySQL es un SGBD desarrollado por la empresa MySQL AB, una empresa de origen sueco que lo desarrolla bajo licencia de código libre (concretamente bajo GPL), aunque también, si se desea, puede ser adquirido con licencia comercial para ser incluido en proyectos no libres.

MySQL es una sistema gestor de base de datos extremadamente rápido. Aunque no ofrece las mismas capacidades y funcionalidades que otras muchas bases de datos, compensa esta pobreza de prestaciones con un rendimiento excelente que hace de ella la base de datos de elección en aquellas situaciones en las que necesitamos sólo unas capacidades básicas.

Las funcionalidades más destacadas de MySQL son:

- Soporte de transacciones (nuevo en MySQL 4.0 si usamos InnoDB como motor de almacenamiento).

- Soporte de replicación (con un *master* actualizando múltiples *slaves*).

- Librería para uso embebido.

- Búsqueda por texto.

- *Cache* de búsquedas (para aumentar el rendimiento).

ANOTACIONES

6.1.3. SAP DB

SAP DB no es más que el SGBD de la empresa de software de gestión empresarial SAP (autores del famoso SAP/R3). Dicha empresa durante mucho tiempo incluía en su *portfolio* de productos un SGBD relacional llamado SAP DB. En abril del 2001 decidieron lanzarlo al mundo con una nueva licencia, la GPL. A partir de ese momento, todo el desarrollo de SAP DB se ha conducido bajo licencia de código libre.

SAP DB es una base de datos muy potente que, por proceder de un entorno muy especializado, el de las aplicaciones de SAP, no está muy extendida por el mundo del código libre. A pesar de ello, SAP DB tiene características muy potentes, lo cual, junto con el gran prestigio de que goza la empresa que la creó, hacen de ella una seria candidata a convertirse en la base de datos de elección para algunos de nuestros proyectos de código libre.

- Soporte de *outer joins*.
- Soporte de roles de usuarios.
- Vistas actualizables.
- Transacciones y bloqueos implícitos.
- Cursores *scrollables*.
- Procedimientos almacenados.

6.1.4. FirebirdSQL

FirebirdSQL es una base de datos de código libre surgida a partir de la versión de código libre de Interbase que Borland/Inprise liberó en el verano de 2000. Como la licencia con la que se liberó dicha versión y el modo de trabajo que tenía previsto seguir Borland no estaban demasiado claros, un grupo de desarrolladores inició su propia versión de Interbase, que se llamó FirebirdSQL.

El primer objetivo que se marcaron los desarrolladores de FirebirdSQL fue estabilizar el código y eliminar multitud de *bugs*, así como aumentar el número de plataformas en las que funcionaba la base de datos. A partir de ese momento, se han desarrollado gradualmente tanto las prestaciones de la base de datos como el número y calidad

ANOTACIONES

de las funciones que ésta ofrece. En la actualidad algunas de las funcionalidades más destacadas son:

- Arquitectura de versiones que evita bloqueos entre lectores y escritores.
- Alerta de eventos para reaccionar a cambios en la base de datos.
- Tipos de datos muy ricos (BLOBS, etc.).
- Procedimientos almacenados y *triggers*.
- Compatibilidad ANSI SQL-92.
- Integridad referencial.
- Transacciones.
- Soporte de múltiples bases de datos interconectadas.

6.2. Controladores y direcciones

6.2.1. Controladores JDBC

A pesar de las muchas similitudes entre los diferentes SGBD, sus lenguajes, prestaciones, etc., los protocolos de comunicación que se deben emplear para acceder a ellos varían totalmente de unos a otros. Por eso, para comunicarnos con los diferentes SGBD desde JDBC deberemos emplear un controlador (un *Driver*) que nos aísle de las peculiaridades del SGBD y de su protocolo de comunicaciones.

Existen diversos tipos de controladores de JDBC, que clasificaremos según el siguiente esquema:

- Controladores de tipo 1. *Bridging Drivers*. Son controladores que traducen las llamadas de JDBC a llamadas de algún otro lengua-

je de acceso a SGBD (por ejemplo, ODBC). Se usan en aquellas situaciones en las que no disponemos de un driver JDBC más adecuado. Implica instalar en la máquina cliente el controlador que nos permite traducir las llamadas JDBC. El más conocido es el driver JDBC-ODBC que actúa de puente entre JDBC y ODBC.

- Controladores de tipo 2. *Native API Partly Java Drivers*. Son controladores que usan el API de Java JNI (Java *native interface*) para presentar una interfase Java a un controlador binario nativo del SGBD. Su uso, igual que los de tipo 1, implica instalar el controlador nativo en la máquina cliente. Suelen tener un rendimiento mejor que los controladores escritos en Java completamente, aunque un error de funcionamiento de la parte nativa del controlador puede causar problemas en la máquina virtual de Java.

- Controladores de tipo 3. *Net-protocol All-Java Drivers*. Controladores escritos en Java que definen un protocolo de comunicaciones que interactúa con un programa de *middleware* que, a su vez, interacciona con un SGBD. El protocolo de comunicaciones con el *middleware* es un protocolo de red independiente del SGBD y el programa de *middleware* debe ser capaz de comunicar los clientes con diversas bases de datos. El inconveniente de esta opción estriba en que debemos tener un nivel más de comunicación y un programa más (el *middleware*).

- Controladores de tipo 4. *Native-protocol All-Java Drivers*. Son los controladores más usados en accesos de tipo intranet (los usados generalmente en aplicaciones web). Son controladores escritos totalmente en Java, que traducen las llamadas JDBC al protocolo de comunicaciones propio del SGBD. No requieren ninguna instalación adicional ni ningún programa extra.

Casi todos los SGBD modernos disponen ya de un controlador JDBC (especialmente de tipo 4).

Nota

Para obtener una lista de los controladores existentes, podemos visitar http://java.sun.com/products/jdbc/jdbc.drivers.html

6.2.2. Cargando el controlador en Java

Para usar un controlador de JDBC, antes deberemos registrarlo en el `DriverManager` de JDBC. Esto se realiza habitualmente cargando

la clase del controlador mediante el método `forName` de la clase `Class`. La construcción habitual es:

```
try
{
 Class.forName("org.postgresql.Driver");
}
catch(ClassNotFoundException e)
{
 ....
}
```

6.2.3. Direcciones de base de datos

Los controladores JDBC para identificar una conexión concreta a una base de datos utilizan un formato de dirección de tipo URL (*Universal Resource Locator*). Esta dirección suele ser de la forma:

```
jdbc:controlador:basededatos
```

El formato en realidad es muy flexible. Los fabricantes tienen total libertad para definirlo.

Algunos de los formatos más usados son:

Tabla 10. Formatos

PostgreSQL	jdbc:postgresql://127.0.0.1:5432/basedatos
Oracle	jdbc:oracle:oci8:@DBHOST
JDBC-ODBC	jdbc:odbc:dsn;opcionesodbc
MySQL	jdbc:mysql://localhost/basedatos?user=jose&password=pepe
SAP DB	jdbc:sapdb://localhost/basedatos

Podemos observar que PostgreSQL especifica la dirección IP del servidor, así como el puerto (5432) y el nombre de la base de datos. Oracle, por otro lado, especifica un sub-controlador (oci8) y un nombre de base de datos que sigue los definidos por Oracle TNS. Podemos ver en los ejemplos de direcciones de conexión que, a pesar de ser diferentes, todas siguen un patrón muy similar (especialmente PostgreSQL, MySQL y SAP DB).

ANOTACIONES

6.2.4. Conectando a la base de datos

El método simple de conexión a una base de datos nos proporcionará un objeto de tipo `Connection` que encapsulará una conexión simple. Podemos tener en cada aplicación tantas conexiones como nos permitan los recursos del sistema (especialmente los del SGBD) y mantener conexiones a diferentes SGBD.

Para obtener una `Connection` emplearemos el método `DriverManager.getConnection()`. Nunca instanciaremos un objeto de tipo `Connection` directamente.

```
Connection con =
   DriverManager.getConnection (“url”,”usuario”,”password”);
```

Pasamos tres parámetros a `getConnection`, la dirección de la base de datos en el formato visto anteriormente, el usuario y la contraseña. Para las bases de datos en las que no es necesario el usuario y la contraseña, los dejaremos en blanco. Cuando llamamos este método, JDBC pregunta a cada controlador registrado si soporta la URL que le pasamos y en caso afirmativo nos devuelve un objeto `Connection`.

Cuando un `Connection` ya no vaya a ser usado más, debemos cerrarlo explícitamente con `close()` para no ocupar recursos. Es especialmente importante liberar las conexiones a bases de datos, ya que constituyen un recurso muy costoso.

JDBC, a partir de la versión 2.0, proporciona además un mecanismo para *pooling* de conexiones, permitiéndonos contar con un bloque de conexiones preestablecidas que, además, se reutilizan de uso en uso.

6.3. Acceso básico a la base de datos

Una vez tenemos un objeto `Connection`, podemos empezar a usarlo para ejecutar comandos SQL en la base de datos. Disponemos de tres tipos básicos de sentencias SQL en JDBC:

`Statement`. Representa una sentencia básica de SQL, sea de consulta (SELECT) o de manipulación de datos (INSERT, UPDATE, etc.).

ANOTACIONES

`PreparedStatement`. Representa una sentencia precompilada de SQL que ofrece mejores prestaciones que las sentencias básicas.

`CallableStatement`. Representa una llamada a un procedimiento almacenado de SQL.

6.3.1. Sentencias básicas

Para obtener un objeto `Statement` utilizaremos el método `createStatemet` del objeto `Connection`:

```
Statement sent=con.createStatement();
```

Creado ya el objeto `Statement`, podemos usarlo para ejecutar comandos SQL en la base de datos. Los comandos SQL pueden o no retornar resultados. Si retornan resultados en forma de tabla (por ejemplo, con un comando SQL de tipo SELECT) utilizaremos el método `executeQuery` de `Statement` para ejecutarlos:

```
ResultSet rs=sent.executeQuery("SELECT * FROM CLIENTES");
```

Estudiaremos `ResultSet` en detalle más adelante. En este código hemos usado `executeQuery` para ejecutar una consulta de datos. Disponemos también de otro método, `executeUpdate`, para ejecutar sentencias que no retornen resultados, por ejemplo UPDATE o DELETE. `executeUpdate` retorna un entero que nos indica qué número de filas se ha visto afectado por el comando SQL enviado.

```
int columnas=sent.executeUpdate("UPDATE CLIENTES SET SALDO=0");
```

Para los casos en que no sabemos *a priori* si una sentencia retornará una tabla de resultados (como `executeQuery`) o un número de filas afectadas (como `executeUpdate`), disponemos finalmente de un método, `execute`, más genérico. `execute` devuelve `true` si hay un `ResultSet` asociado a una sentencia y `false` si dicha sentencia retorna un entero. En el primer caso, podemos recoger el `ResultSet` resultante con `getResultSet`, mientras que en el segundo, mediante `getUpdateCount` podemos recoger el número de filas afectadas.

ANOTACIONES

```
Statement sent=con.createStatement();
if(sent.execute(sentenciaSQL))
{
 ResultSet rs=sent.getResultSet();
 // mostrar resultados
}
else
{
 int afectadas=sent.getUpdateCount();
}
```

Es necesario recordar que un `Statement` representa una única sentencia SQL y, por lo tanto, si hacemos una llamada a `execute`, `executeQuery` o `executeUpdate` los `ResultSet` asociados a ese `Statement` se cierran y liberan. Por lo tanto, es muy importante haber acabado de procesar los `ResultSet` antes de lanzar cualquier otro comando SQL.

Para cerrar un `Statement`, disponemos de un método `close`. A pesar de que al cerrar la `Connection` también se cierran los `Statement` asociados, es mucho mejor cerrarlos explícitamente para poder liberar antes los recursos ocupados.

Múltiples resultados

Es factible que un `Statement` retorne más de un `ResultSet` o más de un número de filas afectadas. `Statement` soporta retornos múltiples con el método `getMoreResults`. Este método retorna `true` si hay más `ResultSet` esperando. Cabe destacar que el método retorna `false` si el siguiente retorno es un número de filas afectadas, a pesar de que pudiese haber más `ResultSet` detrás del número. En este caso sabremos que hemos procesado ya todos los resultados si `getUpdateCount` retorna -1.

Nota

Si una sentencia SQL puede retornar más de un resultado o recuento de columnas modificadas dependerá del SGBD, generalmente a consecuencia de procedimientos almacenados.

Podemos modificar así el código anterior para que soporte múltiples resultados:

```
Statement sent=con.createStatement();
sent.execute(sentenciaSQL);
while(true)
{
 rs=sent.getResultSet();
```

ANOTACIONES

```
 if(rs!=null)
 {
       // mostrar resultados
 }
 else
 {
       // mostrar número
 }
 // Siguiente o final
 if((sent.getMoreResults()==false) &&
       (sent.getUpdateCount()==-1))
       break;
}
```

6.3.2. Resultados

La ejecución de cualquier sentencia SQL de consulta (SELECT) da como resultado una tabla (una pseudo-tabla, en realidad) que contiene los datos que cumplen los criterios establecidos. JDBC utiliza una clase `ResultSet` para encapsular estos resultados y ofrecernos métodos para acceder a ellos.

Podemos imaginarnos `ResultSet` como una serie de datos que nos llegan del SGBD. No podemos retroceder en ella; por lo tanto, debemos procesarlos a medida que avanzamos por la serie. En JDBC 2.0 se ofrecen *scrollable cursors* que nos permiten desplazarnos libremente por los resultados.

Un ejemplo de código que nos permitirá procesar los resultados de un `ResultSet` es el siguiente:

```
Statement sent=con.createStatement();
ResultSet rs=sent.executeQuery("SELECT * FROM CLIENTES");
while(rs.next())
{
  System.out.println("Nombre :"+rs.getString("NOMBRE");
  System.out.println("Ciudad :"+rs.getString("CIUDAD");
}
rs.close();
  sent.close();
```

ANOTACIONES

El código anterior recorre el conjunto de resultados (`ResultSet`) iterando por todas y cada una de las filas con el método `next`. Al principio, una vez obtenido el `ResultSet`, JBDC nos posiciona antes del primer elemento de la lista. Por eso, para acceder al primer elemento debemos llamar a `next`. Para leer la siguiente fila debemos llamar de nuevo `next`. Si no hay más filas que leer, `next` devuelve `false`.

Una vez posicionados en la fila que queremos leer, podemos usar los métodos `getXXXX` para obtener la columna concreta que deseamos visualizar. Disponemos de múltiples métodos `getXXXX`, uno para cada tipo de datos que podemos leer del SGBD. Los métodos `getXXXX` toman como parámetro una cadena que deberá ser el nombre del campo o un parámetro numérico que indicará la columna por posición, teniendo en cuenta que se inicia la numeración en 1, no en 0 como es habitual en `arrays`, etc.

Otro método destacable es `getObject`, que retorna un objeto Java.

Ejemplo

Por ejemplo, si usamos `getObject` con una columna entera, retornará un objeto de tipo `Integer`, mientras que si usamos `getInt`, ésta nos retornará un `int`.

En la tabla siguiente tenemos los tipos de datos de SQL, junto con los tipos de objetos que retorna JDBC y el método propio del tipo. Si el tipo retornado por el método es diferente del objeto Java, dicho tipo va entre paréntesis.

Una opción destacable que podemos usar en múltiples situaciones es usar `getString` para todos los tipos, ya que JDBC realiza la conversión a cadena de caracteres de la mayoría de tipos SQL. Dado que muchas aplicaciones web van a mostrar datos de forma simple, esta opción resulta sumamente interesante.

ANOTACIONES

Tabla 11. Tipos de datos SQL y retornos JDBC

Tipo SQL	Tipo Java	método getXXXX
CHAR	String	getString()
VARCHAR	String	getString()
NUMERIC	java.math.BigDecimal	getBigDecimal()
DECIMAL	java.math.BigDecimal	getBigDecimal()
BIT	Boolean (boolean)	getBoolean()
TINYINT	Integer (byte)	getByte()
SMALLINT	Integer (short)	getShort()
INTEGER	Integer (int)	getInt()
BIGINT	Long (long)	getLong()
REAL	Float (float)	getFloat()
FLOAT	Double (double)	getDouble()
DOUBLE	Double (double)	getDouble()
BINARY	byte[]	getBytes()
VARBINARY	byte[]	getBytes()
LONGVARBINARY	byte[]	getBytes()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Tratamiento de *nulls*

En algunas bases de datos, algunas columnas pueden contener valores nulos (*nulls*). El tratamiento de dichas columnas se complica en JDBC debido a que algunos controladores no lo realizan correctamente. Algunos métodos que retornan objetos retornan `null` en dicho caso, pero son especialmente vulnerables a errores los que no retornan objetos como `getInt`. `getInt`.

Ejemplo

Por ejemplo, si retorna -1 al encontrarse un valor nulo en una columna, evidentemente no podremos distinguir si se trataba de -1 como valor o si la columna era nula.

Disponemos de un método para averiguar si el último resultado obtenido era nulo: `wasNull`.

ANOTACIONES

```
int cantidad=rs.getInt("CANTIDAD");
if(rs.wasNull())
 System.out.println("El resultado era nulo");
else
 System.out.println("Saldo: "+cantidad);
```

Es importante recordar que `wasNull` se refiere a la última columna leída.

Tipos de datos grandes

Podemos obtener *streams* de Java para acceder a columnas que contengan tipos de datos grandes, como imágenes, texto, documentos, etc. Los métodos que proporciona JDBC para ello son `getAsciiStream`, `getBinaryStream` y `getCharacterStream`. Estos métodos retornan un objeto de tipo `InputStream`. El siguiente ejemplo lee un objeto de este tipo de la base de datos y lo escribe en un `OutputStream` que podría corresponder a la salida de un Servlet para mostrar una imagen.

```
ResultSet res=
 sent.executeQuery("SELECT FOTO FROM PERSONAS"+
 " WHERE DNI='"+dni+"'");
if(res.next())
{
 BufferedInputStream imagen=
      new BufferedInputStream
      (res.getBinaryStream("FOTO"));
 byte[] buf=new byte[4096]; // Buffer de 4Kbytes
 int longi;
 while((longi=imagen.read(buf,0,buf.length))!=-1)
 {
      outstream.write(buf,0,len);
 }
}
```

En JDBC 2.0 disponemos además de dos objetos específicos para el tratamiento de objetos grandes (BLOB, *Binary Large Objects* y CLOB, *Character Large Objects*), son `Blob` y `Clob`, respectivamente.

Estos dos objetos nos ofrecen acceder a ellos como `Streams` o directamente con métodos como `getBytes`. Disponemos, además, de métodos para pasar `Blobs` y `Clobs` a sentencias preparadas.

ANOTACIONES

6.3.3. Gestión de errores

Si durante la ejecución un objeto JDBC encuentra un error grave que le impide continuar, normalmente lanzará una excepción, concretamente una `SQLException`.

`SQLException` extiende `Exception` y define diversos métodos adicionales. Uno de estos métodos es `getNextException`. Este método permite encadenar diversas excepciones en una sola, caso de que JDBC encontrase más de un error grave.

`SQLException` define además otros métodos para obtener más información sobre el tipo y la naturaleza del error: `getSQLState` y `getErrorCode`. `GetSQLState` nos devuelve un código de estado/error de la base de datos según la tabla de códigos definidos por ANSI-92 SQL. `getErrorCode` nos devuelve un código de error propio del fabricante del SGBD.

Ejemplo

Por ejemplo URL incorrectas, problemas de seguridad, etc.

Un ejemplo de código completo de un `catch` que gestionase todas las posibles excepciones sería:

```
try
{
      // acciones sobre la base de datos
}
catch (SQLException e)
{
  while(e!=null)
  {
   System.out.println(
        "Excepción: "+e.getMessage());
   System.out.println(
         "Código ANSI-92 SQL: "+e.getSQLState());
   System.out.println(
         "Código del fabricante:"+e.getErrorCode());
   e=e.getNextException();
  }
}
```

Avisos de SQL, SQL *Warnings*

JDBC, además de poder lanzar excepciones en caso de errores, puede generar una serie de avisos de condiciones erróneas pero no graves (que le han permitido continuar). Qué tipo de errores genera un aviso y no un error es una decisión del fabricante de la base de datos y varía de una a otra. Para acceder a dichos avisos, dis-

ANOTACIONES

ponemos del objeto `SQLWarning`, que se usa de una forma parecida a `SQLException`, con la diferencia de que no podemos usarlo en un bloque `try-catch`, sino que debemos interrogar a JDBC si hay `SQLWarnings` después de cada operación.

Durante la fase de depuración (o incluso la de explotación) podemos usar el siguiente "truco" para capturar todos los `SQLWarnings`:

```
void impWarnings(SQLWarning w)
{
 while(w!=null)
 {
     System.out.println("\n SQLWarning: ");
     System.out.println(w.getMessage());
     System.out.println("ANSI-92 SQL Estado: "
           +w.getSQLState());
     System.out.println("Código Fabricante: "
           +w.getErrorCode());
     w=w.getNextWarning();
 }
}
```

Podemos envolver todas las llamadas JDBC con nuestro método `impWarnings` para capturar todas los posibles `SQLWarnings`. Podemos emplearlo como sigue:

```
ResultSet r=sent.executeQuery("SELECT * FROM PROVEEDORES");
impWarnings(sent.getWarnings());
impWarnings(r.getWarnings());
```

6.4. Sentencias preparadas y procedimientos almacenados

Las sentencias estudiadas hasta ahora nos sirven para ejecutar todo tipo de órdenes SQL en la base de datos (sean de inserción, consulta, eliminación, etc.), pero de una forma primitiva. Para acelerar y mejorar el rendimiento, o para ejecutar procedimientos que podamos tener en la base de datos, debemos emplear otros mecanismos.

ANOTACIONES

6.4.1. Sentencias preparadas

Uno de los pasos más costosos en la ejecución de sentencias SQL es la compilación y la planificación de la ejecución de la sentencia. En otras palabras, decisiones sobre qué tablas se consultarán primero, cómo accederemos a ellas, en qué orden, etc. Para evitar este coste, una técnica muy utilizada es la de las sentencias compiladas o preparadas. Las sentencias preparadas son sentencias SQL que se envían al SGBD para que las prepare antes de ejecutarlas y que luego utilizaremos repetidamente cambiando algunos parámetros, pero aprovechando la planificación de una ejecución a la siguiente.

Un objeto `PreparedStatement` se crea, al igual que un `Statement`, a partir de una conexión a la base de datos:

```
PreparedStatement sp=con.prepareStatement(
 "INSERT INTO CLIENTES (ID,NOMBRE) VALUES (?,?)");
```

Como la finalidad del `PreparedStatement` es ejecutar repetidamente una sentencia, no especificamos los valores que tienen que ser insertados, sino que ponemos unos marcadores especiales que posteriormente sustituiremos con los valores que realmente vayamos a insertar:

```
sp.clearParameters();
sp.setString(1,"0298392");
sp.setString(2,"PEPITO GRILLO");
sp.executeUpdate();
```

Antes de adjudicar valores a los marcadores, debemos limpiar la asignación actual. Disponemos de un conjunto de llamadas `setXXXX`, parecidas a las `getXXXX` de `ResultSet`, para adjudicar estos valores. Los valores se referencian posicionalmente empezando por 1. Cabe destacar la llamada `setObject` que nos permite asignar objetos Java a marcadores, encargándose JDBC de la conversión de formato. Disponemos de tres opciones de llamada a `setObject`:

```
setObject(int indice, Object ob, int tipoSQL, int escala);
setObject(int indice, Object ob, int tipoSQL);
setObject(int indice, Object ob);
```

ANOTACIONES

En estas llamadas `tipoSQL` son una referencia numérica a una de las constantes de tipo SQL definidas en la clase `java.sql.Types`.

Para insertar un valor nulo en la base de datos, disponemos de la llamada `setNull` o de la posibilidad de usar `setObject` con un parámetro `null`, eso sí, especificando el tipo SQL que queremos.

6.4.2. Procedimientos almacenados

Muchas bases de datos modernas incluyen un lenguaje de programación propio que permite desarrollar procedimientos y funciones que ejecuta el propio SGBD. Las ventajas de esta aproximación son muchas. Por un lado, tenemos un código independiente de las aplicaciones que puede ser usado desde muchos programas realizados en múltiples lenguajes de programación. Por otro lado, tenemos un código que aísla las aplicaciones del diseño de la base de datos, proporcionando una interfase independiente de la forma de las tablas para realizar algunas operaciones, permitiendo modificar éstas y sólo tener que modificar los procedimientos almacenados. En algunos casos, además, este enfoque representa una mejora sustancial en cuanto a rendimiento, ya que no los datos no han de viajar por la red para ser procesados, sino que todo el proceso se realiza localmente en el SGBD, y posteriormente sólo viajan los resultados.

Ejemplo

Un ejemplo de procedimiento almacenado está escrito en PL/PGSQL, uno de los lenguajes de programación de PostgreSQL; en otros SGBD, el lenguaje y la sintaxis serían totalmente diferentes.

```
CREATE OR REPLACE FUNCTION proNuevoCliente
 (VARCHAR) RETURNS INTEGER AS '
DECLARE
 nombre ALIAS FOR $1;
 iden integer;
BEGIN
 SELECT max(id) INTO iden
```

ANOTACIONES

```
 FROM CLIENTES;
 iden:=iden+1;
INSERT INTO CLIENTES (ID, NOMBRE) VALUES (iden, nombre);
  RETURN iden;
END
' LANGUAGE 'plpgsql';
```

Este ejemplo recibe un parámetro, una cadena con el nombre del cliente, a continuación lo inserta y nos devuelve el identificador asignado a este cliente.

Para llamarlo JDBC nos proporciona el objeto `CallableStatement`. Ya que cada SGBD tiene una sintaxis propia para las llamadas a funciones y procedimientos almacenadas, JDBC nos proporciona una sintaxis estándar para éstas.

Si el procedimiento almacenado no devuelve valores, la sintaxis de la llamada es:

```
{call nombreprocedimiento [(?[,?...])]}
```

Si el procedimiento devuelve valores (como es el caso de la función que hemos definido), entonces la sintaxis es:

```
{? = call nombreprocedimiento [(?[,?...])]}
```

Cabe destacar que los parámetros son opcionales y vienen representados por `?`, igual que en las sentencias preparadas. El controlador JDBC será el responsable de traducir estas llamadas a las correspondientes al SGBD. El código Java que llamaría a la función que hemos definido será:

```
CallableStatement proc=
 con.prepareCall("{?=call proNuevoCliente(?)}");
proc.registerOutParameter(1,Types.INTEGER);
proc.setInt(2,"Nombre");
proc.execute();
System.out.println("Resultado:"+proc.getInteger(1));
```

ANOTACIONES

Sólo es necesario utilizar el `CallableStatement` con los procedimientos almacenados que retornan valores. Podemos llamar a los procedimientos almacenados que no retornan valores mediante los objetos de sentencia anteriores.

6.5. Transacciones

Uno de los aspectos y funcionalidades más importantes que nos ofrecen los SGBD modernos es la posibilidad de realizar transacciones.



Una transacción es un conjunto de operaciones de base de datos que se realizan de forma atómica, es decir, como si fueran una sola operación indivisible. Esto nos permite combinar diversas sentencias SQL para realizar operaciones que nos lleven a un fin concreto.

Ejemplo

Por ejemplo, si insertar un alumno en una base de datos académica supone insertarlo en la tabla de alumnos, crear un registro de expediente, crear uno de correo electrónico, etc., nos interesará garantizar que todas estas operaciones se ejecutan como un conjunto único y atómico.

El trabajo con transacciones implica los siguientes pasos: iniciar la transacción, ejecutar las operaciones, y finalmente, si la transacción es correcta, validarla y grabar los cambios en la base de datos, y si no es correcta o hemos tenido algún problema, deshacer los cambios. Esta capacidad de deshacer los cambios es la clave del funcionamiento de las transacciones, lo cual hace posible que si alguna de las operaciones de nuestra transacción no ha sido correcta, hay que deshacer todos los cambios y dejar la base de datos como si ninguna de las operaciones hubiese tenido lugar. Así, en nuestro caso, es imposible acabar en alguna situación en la que tenemos la entrada en

ANOTACIONES

la tabla de alumnos, pero no la correspondiente entrada en la de expedientes o de correo electrónico.

Otra característica de las transacciones viene dada por la posibilidad de escoger en qué momento los datos objeto de la transacción son visibles para el resto de aplicaciones. Podemos escoger, por ejemplo, que los datos del alumno sólo sean visibles al finalizar completamente la transacción y que no se pueda leer el expediente hasta que no esté realizada toda la transacción.

En JDBC el responsable de la gestión de las transacciones es el objeto `Connection`. Por defecto, JDBC funciona en modo de auto-transacción, es decir, cada operación SQL está dentro de una transacción que es validada inmediatamente. Para realizar múltiples operaciones en una transacción, primero debemos desactivar la validación automática. Para ello disponemos del método `setAutoCommit`. A partir de ese momento, la validación de una transacción la realizaremos con `commit` y la invalidación de ésta con `rollback`.

```
try
 {
 // Desactivamos validación automática
 con.setAutoCommit(false);
 sent.executeUpdate("INSERT
 ...
 sent.executeUpdate("INSERT ....");
 con.commit();
}
catch(SQLException e)
{
 // En caso de error invalidamos
 con.rollback();
}
```

JDBC soporta, además, diversos modos de aislamiento de transacciones que nos permiten controlar cómo la base de datos resuelve los conflictos entre transacciones. JDBC dispone de cinco modos de resolución de transacciones, que pueden no ser soportados por el SGBD. El modo por defecto, el que tendremos activo

ANOTACIONES

si no especificamos nada, dependerá del SGBD. A medida que aumentamos el nivel de aislamiento de transacciones, el rendimiento decae. Los cinco modos definidos en `Connection` son:

- `TRANSACTION_NONE`. Transacciones deshabilitadas o no soportadas.
- `TRANSACTION_READ_UNCOMMITTED`. Transacciones mínimas que permiten *dirty reads*. Las otras transacciones pueden ver los resultados de las operaciones de las transacciones en curso. Si estas transacciones posteriormente invalidan las operaciones, las otras transacciones quedarán con datos inválidos.
- `TRANSACTION_READ_COMMITTED`. Las otras transacciones no pueden ver los datos no validados (*dirty reads* no permitidos).
- `TRANSACTION_REPEATABLE_READ`. Permite lecturas repetibles. Si una transacción lee un dato que posteriormente es alterado por otra (y la modificación validada), si la primera transacción vuelve a leer el dato, no obtendrá un resultado diferente del de la primera vez. Sólo al validar la transacción y volver a iniciarla obtendremos datos diferentes.
- `TRANSACTION_SERIALIZABLE`. Añade al comportamiento de `TRANSACTION_REPEATABLE_READ` la protección contra inserciones. Si una transacción lee de una tabla y otra transacción añade (y valida) datos a la tabla, la primera transacción, en caso de volver a realizar la lectura obtendrá los mismos datos. Obliga al SGBD a tratar las transacciones como si fueran en serie.

6.6. Metadatos

Hasta ahora hemos accedido a los datos que contiene la base de datos. Podemos, además, acceder a otro tipo de datos, los metadatos, que no son sino datos sobre datos. Estos metadatos nos proporcionarán información sobre qué forma tienen los datos con los que tra-

ANOTACIONES

bajamos, así como información sobre las características de la base de datos.

6.6.1. Metadatos de la base de datos

Los metadatos de la base de datos nos van a proporcionar información sobre las características básicas del SGBD que estemos utilizando, así como información sobre algunas características del controlador JDBC que utilicemos. Para proporcionarnos este acceso, Java nos ofrece una clase (DatabaseMetaData), que será la que encapsulará toda esta información.

Obtendremos un objeto de tipos DatabaseMetaData a partir de un objeto Connection. Ello implica que para obtener estos metadatos necesitamos establecer, previamente, una conexión a la base de datos.

```
DatabaseMetaData dbmd= con.getMetaData();
```

Una vez obtenida la instancia de DatabaseMetaData que representará los metadatos de la base de datos, podemos usar los métodos que nos proporciona ésta para acceder a diversos tipos de informaciones sobre la base de datos, el SGBD y el controlador de JDBC. Podemos dividir la información que nos proporciona JDBC en las siguientes categorías:

- Las que obtienen información sobre el SGBD.
- Las que obtienen información sobre el controlador JDBC utilizado.
- Las que obtienen información sobre los límites operativos del SGBD.
- Las que obtienen información sobre el esquema de base de datos.

Información sobre el SGBD

Existen métodos que nos proporcionan algunos datos *informativos* sobre el motor de datos (versiones, fabricante, etc.), así como algunas informaciones que nos pueden ser de utilidad para hacer que nuestros programas reaccionen mejor a posibles cambios del SGBD.

ANOTACIONES

Algunos de estos métodos son:

Tabla 12. Métodos

Nombre	Descripción
getDatabaseProductName	Nombre del SGBD
getDatabaseProductVersion	Versión del SGBD
supportsANSI92SQLEntryLevelSQL	Soporte EL-ANSI-92
supportsANSI92FullSQL	Soporte ANSI-92
supportsGroupBy	Soporte de GROUP BY
supportsMultipleResultSets	Soporte de múltiples resultados
supportsStoredProcedures	Soporte de procedimientos almacenados
supportsTransactions	Soporte de transacciones

Para obtener una lista exhaustiva, podemos consultar la documentación del JDK (Java *Development Kit*).

Información sobre el controlador JDBC utilizado

Al igual que podemos obtener información sobre el SGBD, podemos obtener alguna información sobre el controlador de JDBC. Las informaciones básicas que podemos obtener son nombre y número de versión.

Tabla 13.

Nombre	Descripción
getDriverName	Nombre del controlador
getDriverVersion	Versión del controlador
getDriverMajorVersion	Parte alta de la versión del controlador
getDriverMinorVersion	Parte baja de la versión del controlador

Nota

Algunos métodos de DatabaseMetaData que aceptan cadenas como parámetros aceptan caracteres especiales para las consultas, % para representar cualquier grupo de caracteres y para representar un carácter.

Información sobre los límites operativos del SGBD

Nos proporcionan información sobre los límites del SGBD concreto, la medida máxima de campos, etc. Muy útiles a la hora de adaptar la aplicación para que pueda funcionar de forma independiente respecto al SGBD. Algunos de los mas destacados son:

Tabla 14.

Nombre	Descripción
`getMaxColumnsInSelect`	Número máximo de columnas en una consulta
`getMaxRowSize`	Medida máxima en una fila que permite el SGBD
`getMaxTablesInSelect`	Número máximo de tablas en una consulta.

Información sobre el esquema de base de datos

Podemos también obtener información sobre el esquema (tablas, índices, columnas, etc.) de la base de datos. Esto nos permite realizar un "explorador" de bases de datos que nos mostrará la estructura de la base de datos. Algunos de los métodos son:

Tabla 15.

Nombre	Descripción
`getColumns`	Nos proporciona las columnas de una tabla
`getPrimaryKeys`	Nos proporciona las claves de una tabla
`getTables`	Nos proporciona las tablas de una base de datos
`getSchemas`	Nos proporciona los esquemas de la base de datos

6.6.2. Metadatos de los resultados

No sólo podemos obtener datos sobre el SGBD (`DatabaseMetaData`), sino que además es posible obtener datos sobre los resultados de una consulta, lo cual permite obtener información sobre la estructura de un `ResultSet`. Podemos, de ese modo, conocer el número de columnas, el tipo de éstas y su nombre, además de alguna información adicional (si soporta nulos, etc.). Algunos de los métodos más destacados de `ResultSetMetaData` son:

Tabla 16. Métodos `ResultSet MetaData`

Nombre	Descripción
`getColumnCount`	Número de columnas del `ResultSet`
`getColumnLabel`	Nombre de una columna
`getColumnTypeName`	Nombre del tipo de una columna
`isNullable`	Soporta nulos
`isReadOnly`	Es modificable

ANOTACIONES

Un ejemplo de código que nos muestra la forma de una tabla es:

```
ResultSet rs=sent.executeQuery("SELECT * FROM "+tabla=);
ResultSetMetaData mdr=rs.getMetaData();
int numcolumnas=mdr=getColumnCount();
for(int col=1;col<numcolumnas;col++)
{
 System.out.print(mdr.getColumnLabel(col)+":");
 System.out.println(mdr.getColumnTypeName(col));
}
```

6.7. Práctica: acceso a bases de datos

Crearemos ahora algunos programas simples que nos permitirán acceder a bases de datos desde Java.

El primero realizará la función más simple posible, una consulta:

```
// Ejemplo Simple de JDBC.
// Realiza un select a la BBDD
//

//
import java.sql.*;
public class SelectSimple {
 public static void main(java.lang.String[] args)
  {
     // Carga inicial del driver
    try
    {
      // Escogemos el driver correspondiente a
      // nuestra BBDD
      Class.forName("org.postgresql.Driver");
    }
    catch (ClassNotFoundException e)
    {
      System.out.println("No se ha cargado el driver");
      return;
```

```
    }

   // Todas las operaciones de JDBC deben tratar excepciones

   // de SQL.
   try
    {
       // Conectamos
       Connection con = DriverManager.getConnection
         ("jdbc:postgresql:test", "usuario", "password");
      // Creamos y ejecutamos una sentencia SQL
       Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery
                   ("SELECT * FROM estudiantes");

       // Visualizamos los resultados
       while(rs.next()) {
              System.out.println(
               rs.getString("nombre") + " "+
               rs.getString("apellido1"));
       }

       // Cerramos los recursos de BBDD
       rs.close();
       stmt.close();
       con.close();
    }
    catch (SQLException se)
    {
       // Imprimimos los errores
      System.out.println("SQL Exception: " + se.getMessage());
      se.printStackTrace(System.out);
    }
  }
}
```

ANOTACIONES

El segundo programa realiza una inserción en la base de datos:

```
//
// Ejemplo de actualizacion
//
import java.sql.*;

public class UpdateSimple {
   public static void main(String args[])
   {
      Connection con=null;

         // carga de driver.
         try
         {
              // Escogemos el driver correspondiente
              Class.forName("org.postgresql.Driver");
         }
         catch (ClassNotFoundException e)
         {
              System.out.println("Error en driver");
              return;
         }

      try
      {
      // Conectamos a la BBDD
      con = DriverManager.getConnection
         ("jdbc:postgresql:test", "usuario", "password");


         Statement s = con.createStatement();
         String dni= new String("00000000");
         String nombre= new String("Carles");
         String ape1= new String("Mateu");
         int update_count = s.executeUpdate
              ("INSERT INTO estudiantes (dni,nombre, ape1) " +
              "VALUES('" + dni+ "','" + nombre+ "','" + ape1+ "')");

         System.out.println(update_count + " columnas insertadas.");
         s.close();
      }
      catch( Exception e )
      {

         e.printStackTrace();
      }
      finally
      {
         if( con != null )
          try { con.close(); }
              catch( SQLException e ) { e.printStackTrace(); }

      }
   }
}
```

ANOTACIONES

Y, finalmente, un programa que realiza la consulta usando sentencias preparadas:

```
// Ejemplo de sentencia preparada
//

//
import java.sql.*;

public class SelectPrepared {

       public static void main(java.lang.String[] args)
       {
          if(args.length!=1)
          {
            System.err.println("Argumento: dni del alumno");
            System.exit(1);
          }
          // Carga de driver
          try
          {
            Class.forName("org.postgresql.Driver");
          }
          catch (ClassNotFoundException e)
          {
            System.out.println("Problemas con el driver");
            return;
          }

          try
          {
            // Conectamos a la BBDD.
            Connection con = DriverManager.getConnection
            ("jdbc:postgresql:test", "usuario", "password");

            // Creamos sentencia
            PreparedStatement pstmt;
            String selec="SELECT * FROM estudiantes WHERE dni=?";
            pstmt=con.prepareStatement(selec);

             // Asignamos argumentos
            pstmt.setString(1, args[0]);
            ResultSet rs=pstmt.executeQuery();

             // Pintamos resultados
            while(rs.next()) {
                System.out.println(
                  rs.getString("nombre") + " "+
                  rs.getString("ape1"));
          }

          // Cerramos recursos de BBDD
          rs.close();
          pstmt.close();
          con.close();
         }
          catch (SQLException se)
         {
          // Imprimimos errores
          System.out.println("SQL Exception: " + se.getMessage());
          se.printStackTrace(System.out);
        }
    }
}
```

ANOTACIONES

7. Servicios web

7.1. Introducción a los servicios web

Los servicios web son componentes software que presentan las siguientes características distintivas para el programador:

- Son accesibles a través del protocolo SOAP (*Simple Object Access Protocol*).

- Su interfaz se describe con un documento WSDL (*Web Services Description Language*).

Veremos en detalle qué significan todos esos nombres de protocolos y formatos. SOAP es un protocolo de comunicaciones por paso de mensajes XML que es la base sobre la que sustentan los servicios web. SOAP permite a las aplicaciones enviar mensajes XML a otras aplicaciones. Los mensajes SOAP son unidireccionales, aunque todas las aplicaciones pueden participar como emisoras o receptoras indistintamente. Los mensajes SOAP pueden servir para muchos propósitos: petición/respuesta, mensajería asíncrona, notificación, etc.



SOAP es un protocolo de alto nivel, que sólo define la estructura del mensaje y algunas reglas básicas de procesamiento de éste, siendo completamente independiente del protocolo de transporte. Ello posibilita que los mensajes SOAP puedan ser intercambiados a través de HTTP, SMTP, JMS, etc., donde _HTTP es el más utilizado en estos momentos.

WSDL es un estándar de descripción de servicios web que utiliza para ello un documento XML. Dicho documento proporcionará a las aplicaciones toda la información necesaria para acceder a un servicio

ANOTACIONES

web. El documento describe el propósito del servicio web, sus mecanismos de comunicación, dónde está ubicado, etc.

Otro componente de los servicios web es UDDI (*universal, description, discovery and integration*). Éste es un servicio de registro de servicios web donde éstos se registran almacenando su nombre, la URL de su WSDL, una descripción textual del servicio, etc. Las aplicaciones interesadas pueden consultar, vía SOAP, los servicios registrados en UDDI, para buscar un servicio, etc.

7.2. XML-RPC

XML–RPC es un protocolo de llamada remota a procedimientos que funciona sobre Internet. Es un protocolo mucho más simple que SOAP y mucho más sencillo de implementar. XML–RPC funciona mediante el intercambio de mensajes entre el cliente del servicio y el servidor, usando el protocolo HTTP para el transporte de dichos mensajes. Concretamente, XML–RPC utiliza peticiones POST de HTTP para enviar un mensaje, en formato XML, indicando:

- Procedimiento que se va a ejecutar en el servidor
- Parámetros

El servidor devolverá el resultado en formato XML.

7.2.1. Formato de la petición XML–RPC

Vamos a estudiar el formato de una petición XML–RPC. Para ello partiremos de una petición de ejemplo como la siguiente:

```
POST /RPC2 HTTP/1.1
User-Agent: Frontier/5.1.2
Host: carlesm.com
Content-Type: text/xml
Content-Length: 186

<?xml version="1.0"?>
<methodCall>
```

ANOTACIONES

```
      <methodName>ejemplo.Método</methodName>
      <params>
        <param>
          <value><i4>51</i4></value>
        </param>
      </params>
   </methodCall>
```

Analizaremos ahora esta petición línea a línea. Primero la cabecera del mensaje:

- La `URI` que observamos en primer lugar, `RPC2`, no está definida por el estándar. Ello nos permite, en caso de que el servidor responda a diversos tipos de peticiones, usar dicha `URI` para enrutarlas.

- Los campos `User-Agent` y `Host` son obligatorios y el valor debe ser válido.

- El campo `Content-Type` será siempre `text/xml`.

- El valor de `Content-Length` debe estar siempre presente y ser un valor correcto.

Luego, el cuerpo de éste:

- El mensaje contiene un único elemento `<methodCall>` que contiene los subelementos

- `<methodName>` que contiene la cadena con el nombre del método que se va a invocar.

- Si el método cuenta con parámetros, debe tener un subelemento `<params>`.

- Con tantos `<param>` como parámetros tenga el método.

- Cada uno de los cuales tiene un `<value>`.

ANOTACIONES

Para especificar los posibles valores de los parámetros disponemos de las siguientes marcas que nos permiten especificar escalares:

Tabla 17. Marcas

marca	tipo
`<i4> o <int>`	entero de 4 bytes con signo
`<boolean> 0`	(falso) o 1(cierto)
`<string>`	cadena ASCII
`<double>`	coma flotante con signo y doble precisión
`<dateTime.iso8601>`	día/hora formato iso8601
`<base64>`	binario codificado en base-64

En caso de no especificar ningún tipo, se asignará un tipo <string> por defecto.

Disponemos también de la posibilidad de usar tipos complejos. Para ello disponemos de un tipo <struct>, que presenta la siguiente estructura:

- Contiene una serie de subelementos de tipo <member>.
- Cada uno de estos tiene un <name> y <value> de alguno de los tipos básicos.

Por ejemplo, un parámetro de tipo <struct> sería:

```
<struct>
   <member>
      <name>nombre</name>
      <value><string>Juan Manuel</string></value>
   </member>
   <member>
      <name>Pasaporte</name>
      <value><i4>67821456</i4></value>
   </member>
</struct>
```

Disponemos también de un mecanismo para pasar valores de tipo lista (*array*) a los métodos llamados:

- Contiene un único subelemento de tipo <data>.

ANOTACIONES

- Éste puede contener cualquier número de subelementos <value>.

Por ejemplo:

```
<array>
   <data>
      <value><int>15</int></value>
      <value><string>Hola</string></value>
      <value><boolean>1</boolean></value>
      <value><int>56</int></value>
   </data>
<array>
```

7.2.2. Formato de la respuesta XML–RPC

La respuesta XML–RPC será una respuesta HTTP de tipo 200 (OK) siempre que no haya un error de bajo nivel. Los errores de XML–RPC serán retornados como mensajes de HTTP correctos, notificándose éste en el contenido del mensaje.

La respuesta tendrá el siguiente formato:

- El Content-Type debe ser text/xml.
- El campo Content-Length es obligatorio y deberá ser correcto.
- El cuerpo de la respuesta contendrá un único <methodResponse> con el siguiente formato:
  - Si el proceso ha sido correcto:
    - Contendrá un único campo <params>, el cual
    - contendrá un único campo <param> que, a su vez,
    - contendrá un único campo <value>.
  - En caso de error
    - contendrá un único <fault>, el cual
    - contendrá un único <value> que es un <struct> con los campos
      - faultCode que es <int>, y
      - faultString que es <string>.

ANOTACIONES

Una respuesta correcta de ejemplo sería la siguiente:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 172
Content-Type: text/xml
Date: Fri, 24 Jul 1998 17:26:42 GMT
Server: UserLand Frontier/5.1.2
<?xml version="1.0"?> <methodResponse>
    <params>
       <param>
          <value><string>Hola<string></value>
       </param>
    </params>
</methodResponse>
```

Mientras que una respuesta errónea sería similar a la siguiente:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 444
Content-Type: text/xml
Date: Fri, 24 Jul 1998 17:26:42 GMT
Server: UserLand Frontier/5.1.2
<?xml version="1.0"?> <methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>FaultCode</name>
               <value><int>4</int></value>
            </member>
            <member>
               <name>FaultString</name>
               <value><int>Too many parameters</int></value>
            </member>
         </struct>
      </value>
   </fault>
</methodResponse>
```

ANOTACIONES

7.2.3. Desarrollo de aplicaciones con XML–RPC

En el *kit* de desarrollo de aplicaciones de Java, el JDK, en su versión 1.4.2, no disponemos de soporte para desarrollo de aplicaciones XML–RPC. Por ello debemos usar alguna librería de clases adicional. Emplearemos en este ejemplo la librería de clases del proyecto Apache, disponible en: `http://ws.apache.org/xmlrpc/`.

El código del servidor de ejemplo:

```
import java.util.Hashtable;
import org.apache.xmlrpc.WebServer;

public class JavaServer {

   public Hashtable sumAndDifference(int x, int y) {
      Hashtable result = new Hashtable();
      result.put("sum", new Integer(x + y));
      result.put("difference", new Integer(x - y));
      return result;
   }
   public static void main(String[] args) {
      try {
      WebServer server =new WebServer(9090);
      server.addHandler("sample", new JavaServer());
      } catch (Exception exception) {
      System.err.println("JavaServer:" + exception.toString());
  }
 }
}
```

Como podemos ver, este servidor nos proporciona un método llamado: `sample.sumAndDifference`.

El código de un cliente para el servicio anterior sería:

```
XmlRpcClient server = new XmlRpcClient("192.168.100.1", 9090);

Vector params = new Vector();
params.addElement(new Integer(5));
params.addElement(new Integer(3));

Hashtable result =
      (Hashtable) server.execute(
                                    "sample.sumAndDifference",
                                    params);
int sum = ((Integer) result.get("sum")).intValue();
int difference =
          ((Integer) result.get("difference")).intValue();

System.out.println(
                     "Sum: "
                     + Integer.toString(sum)
                     + ", Difference: "
                     + Integer.toString(difference));
```

ANOTACIONES

ANOTACIONES

7.3. SOAP

SOAP estandariza el intercambio de mensajes entre aplicaciones. Por ello la función básica de SOAP es definir un formato de mensajes estándar (basado en XML) que encapsulará la comunicación entre aplicaciones.

7.3.1. Mensajes SOAP

La forma general de un mensaje SOAP sería:

```
<ENVELOPE atribs>
 <HEADER atribs>
     <directivas />
 </HEADER>
 <BODY atribs>
     <cuerpo />
 </BODY>
 <FAULT atribs>
     <errores />
 </FAULT>
</ENVELOPE>
```

El significado de cada parte será:

- `Envelope`: es el elemento raíz del formato SOAP.
- `Header`: es un elemento opcional, que sirve para extender las funcionalidades básicas de SOAP (seguridad, etc.).
- `Body`: es el elemento que contiene los datos del mensaje. Su presencia es obligatoria.
- `Fault`: en caso de error, esta sección contendrá información sobre la naturaleza de éste.

Existe una especificación ampliada de SOAP llamada SwA (SOAP *with Attachments*) que utiliza la codificación MIME para el transporte de información binaria.

Un mensaje SOAP tiene la siguiente forma, como podemos ver en este mensaje de una llamada a un servicio web concreto de información meteorológica:

```
<?xml version='1.0' encoding='UTF-8'?>
 <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>.
       <ns1:getTiempo
          xmlns:ns1="http://www.uoc.edu/soap-tiempo"
          SOAP-ENV:encodingStyle
          =" http://www.w3.org/2001/09/soap-encoding"
          <codigopostal xsi:type="xsd:string">
          25001
          </codigopostal>
       </ns1:getTiempo>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

En este caso, el mensaje es una petición de servicio (que en el ejemplo es a un método de JavaBeans llamado `getTiempo`). Podemos distinguir dentro del mensaje una parte llamada `ENVELOPE`, que contiene una parte llamada `Body`.

Podemos ver, si observamos el mensaje más detenidamente, que definimos una llamada a un método, llamado `getTiempo`. Estudiando el formato de la llamada:

```
 <ns1:getTiempo
    xmlns:ns1="http://www.uoc.edu/soap-tiempo"
    SOAP-ENV:encodingStyle=
    "http://www.w3.org/2001/09/soap-encoding"
    <codigopostal xsi:type="xsd:string">
    25001
    </codigopostal>
 </ns1:getTiempo>
```

Podemos apreciar que el método recibe un parámetro, llamado `codigopostal` de tipo cadena. SOAP permite definir parámetros de todos los tipos definidos por XSchema además de proporcionarnos algunos tipos propios (como `SOAP:Array`) y de permitirnos definir nuevos tipos.

ANOTACIONES

De forma similar, el método `getTiempo` nos responderá con un mensaje también codificado en SOAP. El mensaje de respuesta será similar a éste:

```
<?xml version='1.0' encoding='UTF-8'?>
 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTiempoResponse
         xmlns:ns1="http://www.uoc.edu/soap-tiempo"
         SOAP-ENV:encodingStyle=" http://www.w3.org/2001/09/soap-encoding
         <tiempoResponse xsi:type="xsd:string">10</tiempoResponse>
      </ns1:getTiempoResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Donde podemos ver en:

```
<ns1:getTiempoResponse
   xmlns:ns1="http://www.uoc.edu/soap-tiempo"
   SOAP-ENV:encodingStyle=" http://www.w3.org/2001/09/soap-encoding
   <tiempoResponse xsi:type="xsd:string">10</tiempoResponse>
</ns1:getTiempoResponse>
```

que el servicio nos responde a la petición de tiempo con el valor de la temperatura. Es habitual, de hecho es un estándar *de facto*, que la estructura que contenga la respuesta se llame como la que contenga la llamada. añadiendo *Response*, en nuestro caso, de `getTiempo` la respuesta nos viene en una estructura `getTiempoResponse`. Además del mensaje en formato XML, SOAP define una extensión de HTTP consistente en una nueva cabecera para la petición de servicio. Dicha cabecera será opcional a partir de la versión 1.2 de SOAP.

7.3.2. Desarrollo de aplicaciones SOAP

SOAP no define ninguna librería estándar que nos permita programar aplicaciones; de hecho, sólo define el formato de mensaje que deben utilizar las aplicaciones y algunas directivas para la comunicación. Para mostrar el desarrollo de un pequeño servicio de web,

ANOTACIONES

utilizaremos una de las librerías SOAP disponibles, en concreto, la Apache AXIS.

La librería Apache AXIS nació como un producto de IBM Alphaworks, y una vez completado el trabajo y lista la librería IBM, la cedió a Apache Foundation, que continuó su desarrollo bajo la licencia Apache. Para usar la librería Apache AXIS, requeriremos, además, de Tomcat (el contenedor de Servlets de Apache) y de Xerces (el analizador de XML de Apache).

Vamos a desarrollar a continuación un pequeño servicio web utilizando Apache AXIS. Utilizaremos unos nombres de métodos y de parámetros que coincidirán con un método real que `XMethods` `(http://www.xmethods.com)` aloja en su servidor y al que podemos acceder libremente.

Definiremos primero una interfaz de Java:

```
public interface IExchange
{
   float getRate( String country1, String country2 );
}
```

Como podemos ver, la interfaz (`IExchange`) sirve para calcular la tasa de cambio entre dos monedas, dados dos países. No es necesaria la implementación del interfaz en caso de Apache AXIS, pero usar interfaces puede ser necesario para otros API de SOAP.

Veremos ahora una implementación “ficticia” de dicha interfaz:

```
public class Exchange implements IExchange
{
public float getRate( String country1, String country2 )
   {
      System.out.println( "getRate( "+ country1 +
        ", "+ country2 + ")" );
      return 0.8551F;
   }
}
```

ANOTACIONES

En este momento ya podemos desplegar el servicio en nuestro Apache AXIS que tenemos ejecutándose. La forma de despliegue dependerá de las librerías que usemos y por ello deberemos consultar éstas. De forma general no requerirán ningún paso adicional a los realizados, siendo ellas las encargadas, en muchas ocasiones, de generar el fichero WSDL, publicar en directorio, etc.

Veremos ahora el código necesario para invocar este servicio:

```
import java.net.*;
import java.util.*;

// Clases relacionadas con el mensaje
import org.apache.soap.*;
// Clases relacionadas con las llamadas
import org.apache.soap.rpc.*;

public class Client
  {
  public static void main( String[] args ) throws Exception
    {
    // Dirección del servicio de Apache SOAP
    URL url = new URL(
    "http://miservidor:8080/soap/servlet/rpcrouter" );

    // Identificador del servicio. Lo hemos proporcionado
    // al desplegar este
    String urn = "urn:demo:cambio";

    // Prepara invocacion del servicio
    Call call = new Call();
    call.setTargetObjectURI( urn );
    call.setMethodName( "getRate" );
    call.setEncodingStyleURI( Constants.NS_URI_SOAP_ENC );

    // Parametros
    Vector params = new Vector();
    params.addElement(
      new Parameter( "country1",
      String.class, "USA", null ) );
    params.addElement(
      new Parameter( "country2",
      String.class, "EUR", null ) );
    call.setParams( params );
```

ANOTACIONES

```
    try
      {
        System.out.println(
        "invocamos service\n"
        + " URL= " + url
        + "\n URN ="
        + urn );

      // invocamos
      Response response = call.invoke( url, "" );
      // Hay fallo?
      if( !response.generatedFault() )
      {
        // NO HAY FALLO
        Parameter result = response.getReturnValue();
        System.out.println( "Result= " + result.getValue() );
        }
      else
        {
        // HAY FALLO
        Fault f = response.getFault();
        System.err.println( "Fault= "
          + f.getFaultCode() + ", "
          + f.getFaultString() );
        }
      }
    // La llamada ha tenido problemas
    catch( SOAPException e )
      {
        System.err.println(
          "SOAPException= " + e.getFaultCode()
          + ", " + e.getMessage() );
      }
    }
}
```

Cambiaremos ahora nuestro programa para conectarnos a un servicio web ya existente. Si visitamos la web de Xmethods, encontraremos un servicio llamado `Currency Exchange Rate`. Anotamos los datos de conexión:

ANOTACIONES

Figura 19.



RPC Profile for Service "Currency Exchange Rate"

As a convenience for those who need to manually configure basic SOAP RPC calls, we provide this page which summarizes all the necessary parameters need to configure a SOAP RPC call.

This information is derived automatically from the service WSDL file. It is a subset of what can be found in the more comprehensive WSDL Analyzer available from the service detail page.

Method Name	getRate	
Endpoint URL	http://services.xmethods.net:80/soap	
SOAPAction		
Method Namespace URI	urn:xmethods-CurrencyExchange	
Input Parameters	country1 country2	string string
Output Parameters	Result	float

Y modificamos en nuestro código la URL y la URN de conexión adecuadamente. A partir de este momento ya podemos acceder al servicio web que está ejecutándose en los ordenadores de `XMethods`. Si deseamos hacer más pruebas en esta web, la de XMethods, disponemos de una lista de servicios web que la comunidad de Internet pone a la disposición del público.

7.4. WSDL y UDDI

WSDL son las siglas de `Web Services Description Language`, un lenguaje, basado en XML que nos permite describir los servicios web que desplegamos. Además, WSDL, también se usa para localizar y ubicar dichos servicios web en la Internet.

Un documento WSDL es en realidad un documento XML que describe algunas características de un servicio web, su localización y ubicación y los métodos y parámetros que soporta.

7.4.1. Estructura de un documento WSDL

Un documento WSDL define un servicio web usando para ello los siguientes elementos XML:

Tabla 18. Elementos XML

Elemento	Significado
`<portType>`	Operaciones proporcionadas por el servicio web
`<message>`	Mensajes usados por el servicio web
`<types>`	Los tipos de datos utilizados por el servicio web
`<binding>`	Los protocolos de comunicaciones usados por el servicio web

Un documento WSDL tiene, pues, una estructura similar a la siguiente:

```
<definitions>
 <types>
    tipos de datos...
 </types>
 <message>
    definiciones del mensaje...
 </message>
 <portType>
    definiciones de operación ...
 </portType>
 <binding>
    definiciones de protocolo...
 </binding>
</definitions>
```

Un documento WSDL puede contener además otros elementos, como extensiones, y un elemento de servicio que hace posible agrupar las diversas definiciones de varios servicios web en un sólo documento WSDL.

Puertos de WSDL

El elemento `<portType>` es el elemento XML de WSDL que define un servicio web, las operaciones que pueden ser realizadas mediante dicho servicio y los mensajes involucrados. El elemento `<portType>` es análogo a una función de biblioteca en pro-

ANOTACIONES

gramación clásica (o a una clase de programación orientada a objetos).

Mensajes WSDL

El elemento `message` define los datos que toman parte en cada operación. Cada mensaje puede consistir en una o diversas partes, y cada parte puede considerarse similar a los parámetros de una llamada de función o método en los lenguajes de programación tradicionales o los orientados a objetos, respectivamente.

Los tipos de datos en WSDL

Disponemos en WSDL del elemento `<types>` para definir los tipos de datos que usaremos en el servicio web. Para dichas definiciones, WSDL hace uso de XML Schema.

Los vínculos en WSDL

El elemento `<binding>` define el formato de mensaje y los detalles de protocolo para cada puerto.

Podemos ver ahora un ejemplo esquemático de cómo es un documento WSDL:

```
<message name="obtTerminoPet">
 <part name="param" type="xs:string"/>
</message>
<message name="obtTerminoResp">
 <part name="valor" type="xs:string"/>
</message>
<portType name="terminosDiccionario">
 <operation name="obtTermino">
    <input message="obtTerminoPet"/>
    <output message="optTerminoResp"/>
 </operation>
</portType>
```

ANOTACIONES

En este ejemplo, el elemento `portType` define `terminosDiccionario` como el nombre de un puerto y `obtTermino` como el nombre de una operación. Dicha operación tiene un mensaje de entrada (parámetro) llamado `obtTerminoPet` y un mensaje de salida (resultado) llamado `obtTerminoResp`. Los dos elementos `message` definen los tipos de datos asociados a los mensajes.

`terminosDiccionario` es el equivalente en programación clásica a una librería de funciones, `obtTermino` es el equivalente a una función y `obtTerminoPet` y `obtTerminoResp` son los equivalentes a los parámetros de entrada y salida, respectivamente.

7.4.2. Puertos

El puerto define el punto de conexión a un servicio web. Puede definirse como una librería de funciones o una clase de la programación clásica o de la orientada a objetos. Cada operación definida para un puerto puede compararse a una función de un lenguaje de programación tradicional.

Tipos de operación

Disponemos de diversos tipos de operación en WSDL. El más habitual es el de petición-respuesta, pero además de éste, disponemos de:

Tabla 19. Tipos de operación en WSDL

Tipo	Descripción
Unidireccional	La operación recibe mensajes pero no retorna respuestas
Petición-respuesta	La operación recibe una petición y devolverá una respuesta
Solicitud-respuesta	La operación puede enviar una petición y esperará la respuesta
Notificación	La operación puede enviar un mensaje pero no espera respuesta

ANOTACIONES

Operaciones unidireccionales

Veamos un ejemplo de operación unidireccional:

```
<message name="nuevoValor">
 <part name="termino" type="xs:string"/>
 <part name="valor" type="xs:string"/>
</message>
<portType name="terminosDiccionario">
 <operation name="nuevoTermino">
    <input name="nuevoTermino" message="nuevoValor"/>
 </operation>
</portType >
```

Como podemos observar, en este ejemplo definimos una operación unidireccional llamada `nuevoTermino`. Esta operación nos permite introducir nuevos términos en nuestro diccionario. Para ello, usamos un mensaje de entrada llamado `nuevoValor` con los parámetros `termino` y `valor`. No obstante, no hemos definido ninguna salida para la operación.

Operaciones de petición-respuesta

Veamos ahora un ejemplo de operación de petición-respuesta:

```
<message name="obtTerminoPet">
 <part name="param" type="xs:string"/>
</message>
<message name="obtTerminoResp">
 <part name="valor" type="xs:string"/>
</message>
<portType name="terminosDiccionario">
 <operation name="obtTermino">
      <input message="obtTerminoPet"/>
      <output message="optTerminoResp"/>
 </operation>
</portType>
```

En este ejemplo, ya visto anteriormente, vemos cómo se definen dos mensajes, uno de entrada y otro de salida, para la operación `obtTermino`.

ANOTACIONES

7.4.3. Enlaces

Los enlaces o vínculos de WSDL (*bindings*) nos permiten definir los formatos de mensaje y de protocolo para los servicios web. Un ejemplo de enlace para una operación de petición-respuesta para SOAP sería:

```
<message name="obtTerminoPet">
   <part name="param" type="xs:string"/>
</message>

<message name="obtTerminoResp">
   <part name="valor" type="xs:string"/>
</message>

<portType name="terminosDiccionario">
   <operation name="obtTermino">
      <input message="obtTerminoPet"/>
      <output message="optTerminoResp"/>
   </operation>
</portType>

<binding type="terminosDiccionario" name="tD">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
<operation>
      <soap:operation
      soapAction="http://uoc.edu/obtTermino"/>
   <input>
      <soap:body use="literal"/>
   </input>
   <output>
      <soap:body use="literal"/>
   </output>
   </operation>
</binding>
```

El elemento `<binding>` tiene dos atributos: el atributo `name` y el atributo `type`. Con el atributo `name` (podemos usar cualquier nombre sin que tenga que ver necesariamente con el nombre usado en la definición de *port*) definimos el nombre del vínculo y el atributo `type` apunta al puerto del vínculo; en este caso el puerto `terminosDiccionario`.

El elemento `soap:binding` tiene dos atributos `style` y `transport`.

El atributo `style` puede ser `rpc` o `document`. En nuestro ejemplo hemos usado `document`. El atributo `transport` define qué protocolo SOAP usar, en este caso HTTP.

ANOTACIONES

El elemento `operation` define cada una de las operaciones que proporciona el puerto. Para cada uno debemos especificar la acción SOAP correspondiente. Debemos especificar, además, cómo se codificarán la entrada (`input`) y la salida (`output`). En nuestro caso, la codificación es "`literal`".

7.4.4. UDDI

UDDI son las siglas de *Universal Description, Discovery and Integration*, un servicio de directorio donde las empresas y usuarios pueden publicar y buscar servicios web. UDDI es una estructura estándar e independiente de plataforma para describir estos servicios web, buscar servicios, etc.

UDDI está construido sobre los estándares de Internet del W3C y de la IETF (Internet *Engineering Task Force*), como XML, HTTP, etc. Para describir las interfaces hacia los servicios web, utiliza el lenguaje WSDL descrito anteriormente y para las necesidades de programación multiplataforma utiliza SOAP, cosa que permite una total interoperabilidad.

UDDI representa un gran avance para el desarrollo de Internet como plataforma de negocios de tecnologías de la información. Antes de su desarrollo no existía ningún estándar que permitiese localizar, ubicar o publicitar servicios de tratamiento de información. Tampoco existía un método que pudiera integrar los diversos sistemas de información de las compañías.

Algunos de los beneficios que se derivan del uso de UDDI son:

- Permite descubrir el negocio (o servicio) adecuado de los miles que están registrados actualmente en algunos servidores vía Internet.

- Define cómo interactuar con el servicio escogido, una vez localizado éste.

- Nos permite llegar a nuevos clientes y facilita y simplifica el acceso a los clientes ya existentes.

ANOTACIONES

- Extiende el mercado de usuarios potencial de nuestros métodos de negocio y servicios.

- Describe los servicios y componentes o métodos de negocio de forma automática (o automatizable) en un entorno seguro, abierto y simple.

Uso de UDDI

Veamos un posible ejemplo de cómo se podría usar UDDI para solucionar un caso hipotético, pero que nos mostrará claramente las ventajas de UDDI.

Ejemplo

Si la industria hotelera publicase un estándar UDDI para la reserva de habitaciones, las diferentes cadenas hoteleras podrían registrar sus servicios en un directorio UDDI. Las agencias de viajes podrían entonces buscar en el directorio UDDI para encontrar la interfase de reservas de la cadena hotelera. Cuando se encuentre la interfase, la agencia de viajes podría realizar la reserva, ya que dicho interfase estaría descrito en un estándar conocido.

Programación en UDDI

Disponemos de un par de API para el desarrollo de aplicaciones usando UDDI. La mayor parte de implementaciones existentes para desarrollar usando UDDI son comerciales, pero existen dos implementaciones de código libre: pUDDing (`http://www.opensorcerer.org/`) y jUDDI (`http://ws.apache.org/juddi/`). Los dos API mencionados son el API de consulta (*inquiry*) y el de publicación (*publish*). El API *inquiry* busca información sobre los servicios ofrecidos por una empresa, la especificación de éstos e información sobre qué hacer en caso de error. Todos los accesos de lectura a los registros de UDDI usan mensajes del API de *inquiry*. Éste, además, no requiere autenticación, por lo que se usa HTTP para el acceso.

ANOTACIONES

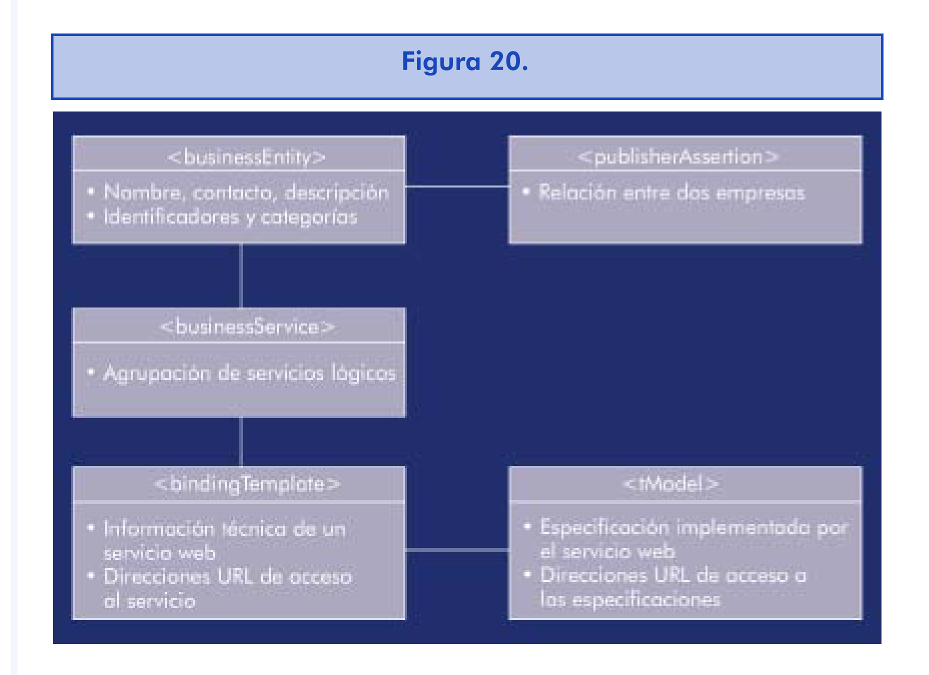
El otro API, el de publicación, es el utilizado para la creación, registro, actualización, etc., de información ubicada en un registro de UDDI. Todas las funciones en este API requieren acceso autenticado a un registro UDDI, por lo que en lugar de usar HTTP se usa HTTPS.

Ambos API están diseñados para ser simples. Todas las operaciones son síncronas y tienen un funcionamiento sin estado.

Para comprender mejor la estructura de los API, el siguiente esquema muestra la relación existente entre las diversas estructuras de datos de UDDI.

Podemos ver un sencillo ejemplo de cómo un cliente enviaría una petición UDDI para encontrar un negocio.

Figura 20.



Para buscar el negocio, la petición que enviaríamos sería:

```
<uddi:find_business generic="2.0" maxRows="10">
 <uddi:name>
     Empresa de software
 </uddi:name>
</uddi:find_business>
```

ANOTACIONES

Como podemos ver, el mensaje que hay que enviar es muy simple, indicándole tan sólo el nombre del negocio. La respuesta que recibiríamos a una petición así sería:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
     <businessList xmlns="urn:uddi-org:api_v2"
        generic="2.0" operator="SYSTINET">
        <businessInfos>
         <businessInfo
         businessKey="132befd0-d09a-b788-ad82-987878dead98">
         <name xml:lang="es">
           Empresa de Software
         </name>
         <description xml:lang="es">
              Una empresa que desarrolla software de servicios web
         </description>
         <serviceInfos>
          <serviceInfo
            serviceKey="23846ac0-dd99-22e3-80a9-801eef988989"
            businessKey="9a26b6e0-c15f-11d5-85a3-801eef208714">
                 <name xml:lang="es">
                       Envia
                 </name>
          </serviceInfo>
         </serviceInfos>
        </businessInfo>
        </businessInfos>
     </businessList>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nota

UUID.

Las instancias de las estructuras de datos están identificadas mediante un identificador único universal, conocido como UUID. Los UUID son asignados cuando la estructura se inserta por primera vez en el registro. Son cadenas hexadecimales estructuradas según el estándar: ISO/IEC 11578:1996, cuya unicidad queda garantizada.

7.5. Seguridad

La aparición de los servicios web ha supuesto la emergencia de algunas cuestiones que hasta ahora no se habían tomado en cuenta. El alto nivel de interoperabilidad y la facilidad de conexión e intercambio de datos que permiten los servicios web supone que aparecen nuevos riesgos en la seguridad de los datos no existentes hasta el momento:

- La claridad de la codificación XML (que permite que sea entendible sin dificultades) facilita en demasía el trabajo a un "enemigo" como un *hacker*, etc.

ANOTACIONES

- Es necesario definir un estándar de interoperabilidad, ya que los diversos participantes en una transacción usando servicios web no tienen por qué usar el mismo software del fabricante.

- La misma naturaleza de SOAP/WS que facilita la aparición de uniones esporádicas entre máquinas, la aparición de intermediarios (*proxies*, etc.) dificulta el control sobre quién tiene acceso a los datos enviados.

- Las aplicaciones ahora están disponibles para todo el mundo, mediante el uso de protocolos comunes, como http; universalizamos el acceso a nuestras aplicaciones. E incluso con estándares como WSDL y UDDI existen directorios de aplicaciones donde todos puede descubrir las aplicaciones publicadas, sus métodos, etc.

Por eso, en la definición de los servicios web está cobrando cada vez más importancia la definición de unos mecanismos de seguridad diseñados específicamente para los servicios web que vayan más allá de SSL.

7.5.1. Incorporación de mecanismos de seguridad en XML

Los actuales estándares, que en el momento de escribir estas líneas aún están en desarrollo, nos permiten ofrecer algunas prestaciones básicas de seguridad:

- Firma digital, que nos permitirá comprobar la identidad del emisor de un mensaje, así como la integridad de éste.

- Autenticación, permitiéndonos verificar identidades.

- No repudio, permitiéndonos evitar que un emisor niegue ser el origen de un mensaje.

Para ello, será necesario disponer de una infraestructura de clave pública operativa (PKI).

ANOTACIONES

Firma digital

La firma digital es un equivalente matemático a la firma manuscrita. Consiste en un código que se añade a un bloque de información y que nos permite verificar el origen de éste y la integridad del mismo.

La especificación de firma digital de XML define un elemento opcional que facilita la inclusión de una firma digital en un documento XML.

Un ejemplo de documento firmado con XML es:

```
<Notas xmlns="urn:uocedu">
   <alumno id="1293">
      <nombre>Joan Oro</nombre>
        <ciudad>Lleida</ciudad>
   </alumno>
   <notas>
      <nota id="BIOMOL">
         <asignatura>Biologia Molecular</asignatura>
         <calif>9.56</calif>
         <comentario>Excelente trabajo</comentario>
      </nota>
   </notas>
   <Signature Id="EnvelopedSig"
      xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo Id="EnvelopedSig.SigInfo">
      <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference Id="EnvelopedSig.Ref" URI="">
           <Transforms>
           <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>
            yHIsORnxE3nAObbjMKVo1qEbToQ=
            </DigestValue>
         </Reference>
```

ANOTACIONES

```
        </SignedInfo>
        <SignatureValue Id="EnvelopedSig.SigValue">
GqWAmNzBCXrogn0BlC2VJYA8CS7gu9xH/XVWFa08eY9HqVnr
fU6Eh5Ig6wlcvj4RrpxnNklBnOuvvJCKqllQy4e76Tduvq/N
8kVd0SkYf2QZAC+j1IqUPFQe8CNA0CfUrHZdiS4TDDVv4sf0
V1c6UBj7zT7leCQxAdgpOg/2Cxc=
        </SignatureValue>
        <KeyInfo Id="EnvelopedSig.KeyInfo">
         <KeyValue>
            <RSAKeyValue>
        <Modulus>
AIvPY8i2eRs9C5FRc61PAOtQ5fM+g3R1Yr6mJVd5zFrRRrJz B/
awFLXb73kSlWqHao+3nxuF38rRkqiQ0HmqgsoKgWChXmLu
Q5RqKJi1qxOG+WoTvdYY/KB2q9mTDj0X8+OGlkSCZPRTkGIK
jD7rw4Vvml7nKlqWg/NhCLWCQFWZ
         </Modulus>
         <Exponent>
         AQAB
         </Exponent>
      </RSAKeyValue>
    </KeyValue>
  </KeyInfo>
 </Signature>
</Notas>
```

Como podemos ver en el ejemplo, la firma digital se aplica encriptando con una clave privada un *digest* (resumen) del mensaje XML. El resumen suele ser el resultado de aplicar una función matemática, llamada *hash*, al documento XML que queremos firmar. Incluimos dicho resumen cifrado junto con una

clave que permite verificar la operación con el mensaje XML. Como sólo nosotros disponemos de la clave que permite cifrar, queda patente que somos los únicos emisores del mensaje. Por otro lado, como el resultado del resumen (*digest*) depende de todos y cada uno de los bytes que componen el mensaje, queda patente que éste no ha sido alterado en curso.

Conviene destacar que, en el caso de XML, dada su peculiar naturaleza y el tratamiento al que suele someterse un mensaje de un servicio web durante su existencia y teniendo en cuenta el análisis (*parseo*), etc., que puede alterar su forma (un cambio de un sólo espacio resulta en un *digest* diferente), la implementación de los *digest* debe tener en cuenta estas alteraciones válidas. Por eso, antes de

ANOTACIONES

calcular el *digest* se debe realizar un proceso que asegurará que ninguno de los posibles cambios que pueda sufrir nuestro XML y que sea un cambio que no afecte realmente el contenido (eliminación de espacios, resumido de *tags* sin contenido, etc.) no signifique la no validación del *digest*. Este proceso, llamado canonicalización W3C-XML-Canonicalization (xml-c14n), consta, entre otras, de las siguientes reglas que debe cumplir un documento antes de calcular el *digest*:

- La codificación debe ser UTF-8.
- Normalización de los saltos de línea (al ASCII 10).
- Normalización de los atributos.
- Elementos vacíos convertidos en pares inicio-fin.
- Normalización de espacios sin significado.
- Eliminación de declaraciones de espacios de nombres superfluas.
- Los atributos por defecto son explicitados.
- Reordenación lexicográfica de las declaraciones y atributos.

Encriptación de los datos

Además de proporcionarnos capacidades de firma digital, los nuevos estándares de seguridad de XML nos proporcionan capacidades de encriptación, que nos permiten ocultar partes o todo el documento XML de los elementos de proceso intermedio.

Un ejemplo nos mostrará cómo quedaría el documento anterior si además de firmar encriptásemos los datos de las notas del alumno.

```
<Notas xmlns="urn:uocedu">
   <alumno id="1293">
      <nombre>Joan Oro</nombre>
      <ciudad>Lleida</ciudad>
   </alumno>
   <notas>
   <EncryptedData Id="ED" Nonce="16"
     Type=http://www.w3.org/2001/04/xmlenc#Content
     xmlns="http://www.w3.org/2001/04/xmlenc#"
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <EncryptionMethod
     Algorithm ="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
```

ANOTACIONES

```
          <ds:KeyInfo>
               <ds:KeyName>uoc</ds:KeyName>
          </ds:KeyInfo>
          <CipherData>
               <CipherValue>
dRDdYjYs11jW5EDy0lucPkWsBB3NmK0AFNxvFjfeUKxP75
cx7KP0PB3BjXPg14kJv74i7F00XZ5WhqOISswIkdN/pIVe
qRZWqOVjFA8izR6wqOb7UCpH+weoGt0UFOEkIDGbemm23e
u812Ob5eYVL8n/DtO81OhYeCXksSMGUZiUNj/tfBCAjvqG
2jlslQM6n4jJ3QNaR4+B2RisOD6Ln+x2UtNu2J7wIYmlUe
7mSgZiJ5eHym8EpkE4vjmr2oCWwTUu91xcayZtbEpOFVFs
6A==
          </CipherValue>
         </CipherData>
   </EncryptedData>
   </notas>
   <Signature Id="EnvelopedSig"
      xmlns="http://www.w3.org/2000/09/xmldsig#">
         .....
   </Signature>
</Notas>
```

8. Utilización y mantenimiento

8.1. Configuración de opciones de seguridad

8.1.1. Autenticación de usuarios

La autenticación de usuarios en Apache nos permite asegurar que las personas que visualizan o acceden a ciertos recursos son aquellos que nosotros deseamos que sean, o bien que conocen una determinada "clave" de acceso. Apache nos permite definir qué recursos precisan de una autenticación del usuario que accede a ellos, además de identificar el mecanismo por el que podemos autenticar a éste.

Nota

Autenticación. El punto débil de la autenticación siempre es el factor humano. Es habitual "dejar" nuestro usuario y contraseña a un compañero para *agilizar* el trabajo.

El nombre de los módulos de autenticación de usuarios suele empezar por `mod_auth_` seguido de una abreviación del mecanismo de búsqueda y verificación de los usuarios, por ejemplo, `ldap`, `md5`, etc.

Mostraremos ahora cómo configurar Apache para requerir autenticación de usuario a fin de acceder a un directorio concreto:

```
<Directory /web/www.uoc.edu/docs/secreto>
   AuthType Basic
   AuthName "Restringido"
   AuthUserFile /home/carlesm/apache/passwd/passwords
   Require user carlesm
</Directory>
```

Con esta configuración le estamos indicando a Apache que sólo muestre los contenidos especificados al usuario `carlesm` después de verificar la identidad de éste mediante una contraseña. El fichero que contendrá los nombres de usuario y las contraseñas es el especificado mediante `AuthUserFile`.

Esta configuración utiliza el módulo `mod_auth`, el cual proporciona una autenticación básica, basada en disponer de un fichero de texto plano donde almacenaremos una lista de usuarios y de palabras de

ANOTACIONES

acceso cifradas (password). Apache nos proporciona además una herramienta para administrar dicho fichero de palabras clave (htpasswd). Para crear un fichero como el que requerimos, poniendo una clave a nuestro usuario, ejecutaremos:

```
# htpasswd -c /home/carlesm/apache/passwd/passwords carlesm
New password: <clave>
Re-type new password: <clave>
Adding password for user carlesm
```

Donde ponemos <clave> debemos teclear nuestra contraseña.

Si continuamos examinando la configuración que hemos proporcionado para la autenticación, veremos que además del fichero de claves requerido, indicamos a Apache el nombre de la zona de autenticación, en nuestro caso, "Restringido". Con ello, el navegador del cliente enviará siempre la clave que nosotros proporcionemos a las peticiones que se identifiquen (la identificación se conoce como Realm) como "Restringido". Ello nos permite compartir palabras clave entre diversos recursos, ahorrándole al usuario la necesidad de teclear demasiadas claves.

Otros módulos de autenticación

Existen múltiples módulos de autenticación para Apache. Estos módulos nos permiten seleccionar múltiples fuentes para verificar los usuarios y las claves. En nuestro ejemplo, hemos usado el módulo básico mod_auth, que proporciona un fichero de texto plano con los usuarios y sus claves, usando la directiva AuthUserFile para indicar el fichero.

Disponemos también de los siguientes módulos:

- mod_auth_dbm: permite el uso de bases de datos tipo Berkeley (DBM) para almacenar los usuarios y sus claves.
- mod_auth_digest: similar a mod_auth, pero permite el uso de DigestMD5 para encriptar las claves.
- mod_auth_ldap: permite el uso de un directorio LDAP para el almacenamiento de usuarios y claves.

ANOTACIONES

- `mod_auth_samba`: permite usar un servidor de dominio Samba (SMB/-CIFS)
- para verificación de usuarios.
- `mod_auth_mysql` y `mod_auth_pgsql`: nos permite almacenar los usuarios y las claves en una base de datos SQL como `mysql` o `postgresql`.
- `mod_auth_radius`: permite usar un servidor RADIUS para verificar los usuarios y las claves.

8.1.2. Seguridad de comunicaciones

Otra de las capacidades de Apache es la de usar criptografía fuerte para cifrar y firmar las comunicaciones. Para ello se sirve de un módulo, llamado `mod_ssl`, que sirve de interfaz hacia la librería criptográfica OpenSSL, proporcionando así a Apache los mecanismos para usar conexiones seguras según el protocolo SSL/TLS.

Para usar el módulo criptográfico debemos instalar `mod_ssl` (para ello previamente deberemos tener instalado OpenSSL en nuestro sistema). Una vez instalado, debemos configurar en el fichero de configuración de Apache el funcionamiento de `mod_ssl`. Para ello, especificaremos los requerimientos de SSL en un servidor virtual o en un directorio:

```
# Activar SSL
SSLEngine on
# Especificar Protocolo
SSLProtocol all -SSLv3
# Especificamos criptoalgoritmos
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
# Ficheros de claves y certificados
SSLCertificateFile /etc/httpd/ssl.key/server.crt
SSLCertificateKeyFile /etc/httpd/ssl.key/server.key
```

Como podemos ver, para la activación de un servidor seguro con SSL hace falta disponer de un par de claves pública/privada, ade-

más de un certificado digital. Para crear certificados y pares de claves podemos usar los programas proporcionados con la librería OpenSSL.

8.2. Configuración de balanceo de carga

Apache provee de diversas opciones para utilizar una solución de balanceo de carga. Las soluciones presentan grados de complejidad muy variables y prestaciones muy diversas, haciendo que las situaciones a las que van encaminadas sean muy distintas.

8.2.1. Balanceo basado en DNS

La solución más simple a un balanceo de carga consiste en instalar un grupo de sistemas con los servidores web y haciendo que todos tengan acceso a los ficheros que forman nuestra web. Entonces podemos programar el servidor de nombres (el DNS) para que devuelva una dirección distinta (de una de las máquinas) cada vez que se le interroga por el nombre de nuestro servidor web.

Nota

El **DNS** (*Domain Name System*) es el servicio encargado, a petición de los clientes, de "resolver" (convertir) los nombres de servidores a direcciones IP. Por ejemplo, el DNS es el responsable de convertir `www.uoc.edu` en 213.73.40.217.

Ejemplo

Así, disponemos de siete ordenadores para configurar nuestro servidor web (www.uoc.edu). Asignamos a cada uno una dirección IP y un nombre (lo que mostramos aquí es el formato de datos de BIND, el programario de DNS más usado):

```
www0    IN    A    1.2.3.1
www1    IN    A    1.2.3.2
www2    IN    A    1.2.3.3
www3    IN    A    1.2.3.4
www4    IN    A    1.2.3.5
www5    IN    A    1.2.3.6
www6    IN    A    1.2.3.7
```

ANOTACIONES

Además, añadimos la siguiente entrada:

```
www       IN        CNAME          www0.uoc.edu.
          IN        CNAME          www1.uoc.edu.
          IN        CNAME          www2.uoc.edu.
          IN        CNAME          www3.uoc.edu.
          IN        CNAME          www4.uoc.edu.
          IN        CNAME          www5.uoc.edu.
          IN        CNAME          www6.uoc.edu.
```

Si ahora pedimos al servidor de nombres que nos indique la dirección de `www.uoc.edu`, éste nos responderá cada vez con todas las direcciones de las siete máquinas, pero en un orden diferente. Esto provocará que las peticiones de los clientes se vayan alternando entre las diferentes máquinas servidoras.

De todas formas, este esquema de balanceo no es un sistema perfecto. Por un lado, no tiene en cuenta si alguna de las máquinas no está funcionando. Por otro lado, el sistema global de DNS obliga a que los servidores de DNS, por ejemplo, el de nuestro proveedor de Internet, hagan *cache* de las direcciones, con lo que algunos servidores de DNS se "aprenderán" una de las direcciones y siempre resolverán esa misma dirección. A largo plazo, este comportamiento no supondrá un problema, pues el total de las consultas se divide entre los diferentes servidores.

Existen otros programas servidores de DNS que ofrecen opciones más efectivas para balanceo de carga real entre servidores, pues utilizan herramientas auxiliares para comprobar el estado de los diversos servidores.

8.2.2. Balanceo basado en Proxy

Otra opción para realizar balanceo para Apache consiste en utilizar un módulo llamado `mod_rewrite` para que redistribuya las peticiones entre los diferentes servidores.

Para ello, modificaremos el servidor de nombres para que nuestra web, llamada `www.uoc.edu`, corresponda a una sola de las máquinas (`www0.uoc.edu`).

ANOTACIONES

```
www        IN         CNAME      www0.uoc.edu.
```

Entonces convertiremos www0.uoc.edu en un servidor proxy dedicado. Con ello conseguiremos que esta máquina reenvíe todas las peticiones que reciba hacia una de las otras cinco máquinas. Para ello configuraremos el módulo de mod_rewrite de manera que reenvíe las peticiones a las otras máquinas de forma balanceada. Utilizamos un programa (en este caso, escrito en Perl) que proporcionará a mod_rewrite el servidor donde debe redireccionar la petición.

```
RewriteEngine on
RewriteMap     lb         prg:/programas/lb.pl
RewriteRule    ^/(.+)$    ${lb:$1}              [P,L]
```

El programa en cuestión es el siguiente:

```
#!/usr/bin/perl
##
## lb.pl -- Balanceo de carga
##
$ = 1;
 $nombre = "www";           # Base de nombre
 $prim = 1;                 # Primer servidor (empezamos por www1)
 $ult = 5;                  # Último servidor (www5).
 $dominio = "uoc.edu"; # Dominio
 $cnt = 0;
 while (<STDIN>)
 {
    $cnt = (($cnt+1) % ($ult+1-$prim));
    $servidor = sprintf(" %s %d. %s", $nombre, $cnt+$prim, $dominio);
    print "http://$servidor/$_";
 }
```

Esta solución presenta el ligero inconveniente de que todas las peticiones pasan por una máquina (www0.uoc.edu). A pesar de que dicha máquina no realiza ningún proceso pesado (CGI, etc.) y sólo realiza reenvíos de peticiones (mucho más rápidos y ligeros), en caso de sobrecarga, esta máquina puede saturarse. Para solucionar este punto disponemos de soluciones *hardware* que realizan las mismas funciones que la receta presentada aquí con mod_rewrite. Dichas soluciones *hardware* suelen ser, eso sí, realmente caras.

8.2.3. Balanceo basado en mod_backhand

Para aquellas situaciones donde el balanceo basado en DNS o en un recurso *hardware* no es suficiente, disponemos de un mecanismo mucho más sofisticado en la asignación de recursos. Dicho mecanismo consiste en un módulo de Apache llamado `mod_backhand` que permite redireccionar las peticiones

HTTP de un servidor web a otro, en función de la carga y la utilización de recursos en los servidores.

`mod_backhand` presenta un inconveniente muy grave: en el momento de escribir estas líneas, no existe aún una versión para Apache 2, debiendo usarse sólo con Apache 1.2/1.3.

La configuración de `mod_backhand` es realmente simple. El módulo se debe instalar en cada uno de los servidores Apache del clúster de máquinas. A continuación, debemos configurar los diferentes servidores Apache para que se comuniquen entre sí con el fin de sincronizarse. Usaremos una dirección de *multicast* cualquiera (239.255.221.20) para comunicarnos. La configuración será similar a:

```
<IfModule mod_backhand.c>
 # Directorio de trabajo de mod_backhand. Debemos asegurarnos de
 # que los permisos nos permiten escribir.UnixSocketDir /var/backhand/backhand
 #   Usaremos IP Multicast con TTL=1 para reportar estadísticas
 #   Podríamos usar Broadcast.
   MulticastStats 239.255.221.20:4445,1
 #   Aceptaremos notificaciones de nuestros servidores:
   AcceptStats 1.2.3.0/24
</IfModule>
```

Podemos configurar `mod_backhand` para que nos muestre en una página el estado de funcionamiento:

```
<Location "/backhand/">
     SetHandler backhand-handler
</Location>
```

ANOTACIONES

Si deseamos activar `mod_backhand` para un directorio, podemos configurarlo en el interior de una sección `Directory` de Apache. Indicaremos a continuación que el directorio `cgi-bin`, que contiene los ficheros CGI compartidos, los de mayor requerimiento de sistema, se halla distribuido entre las máquinas del clúster:

```
<Directory "/var/backhand/cgi-bin">
      AllowOverride None
      Options None
      Order allow,deny
      Allow from all
      Backhand byAge
      Backhand byRandom
      Backhand byLogWindow
      Backhand byLoad
</Directory>
```

Ejemplo

Ejemplo de un criterio para reordenar: por menos carga de CPU.

Como podemos ver, `mod_backhand` se configura mediante unas directivas `Backhand` que denominaremos *funciones de candidaturas*. El funcionamiento de `mod_backhand` es el siguiente: cuando tenemos que servir un recurso del directorio especificado `/var/backhand/cgi-bin`, pasamos a `mod_backhand` la lista de servidores candidatos (inicialmente, todos los configurados con `mod_backhand`). Éste va pasando a cada una de las funciones candidatas especificadas (`byAge`, `byRandom`, etc.). Estas funciones eliminan de la lista los servidores que no consideren adecuados, o bien reordenan la lista según el criterio que corresponda.

A continuación, al acabar de evaluar todas las funciones candidatas, `mod_backhand` reenvía la petición al servidor que ha quedado como primero de la lista de candidatos.

`mod_backhand` dispone de una gran cantidad de funciones candidatas por defecto e incluye también las herramientas para construir nosotros mismos nuevas funciones.

8.2.4. Balanceo utilizando LVS

Los servidores Linux disponen de una herramienta de alto rendimiento para realizar configuraciones de balanceo de carga y de alta disponibilidad. Dicha herramienta (o más propiamente conjunto de

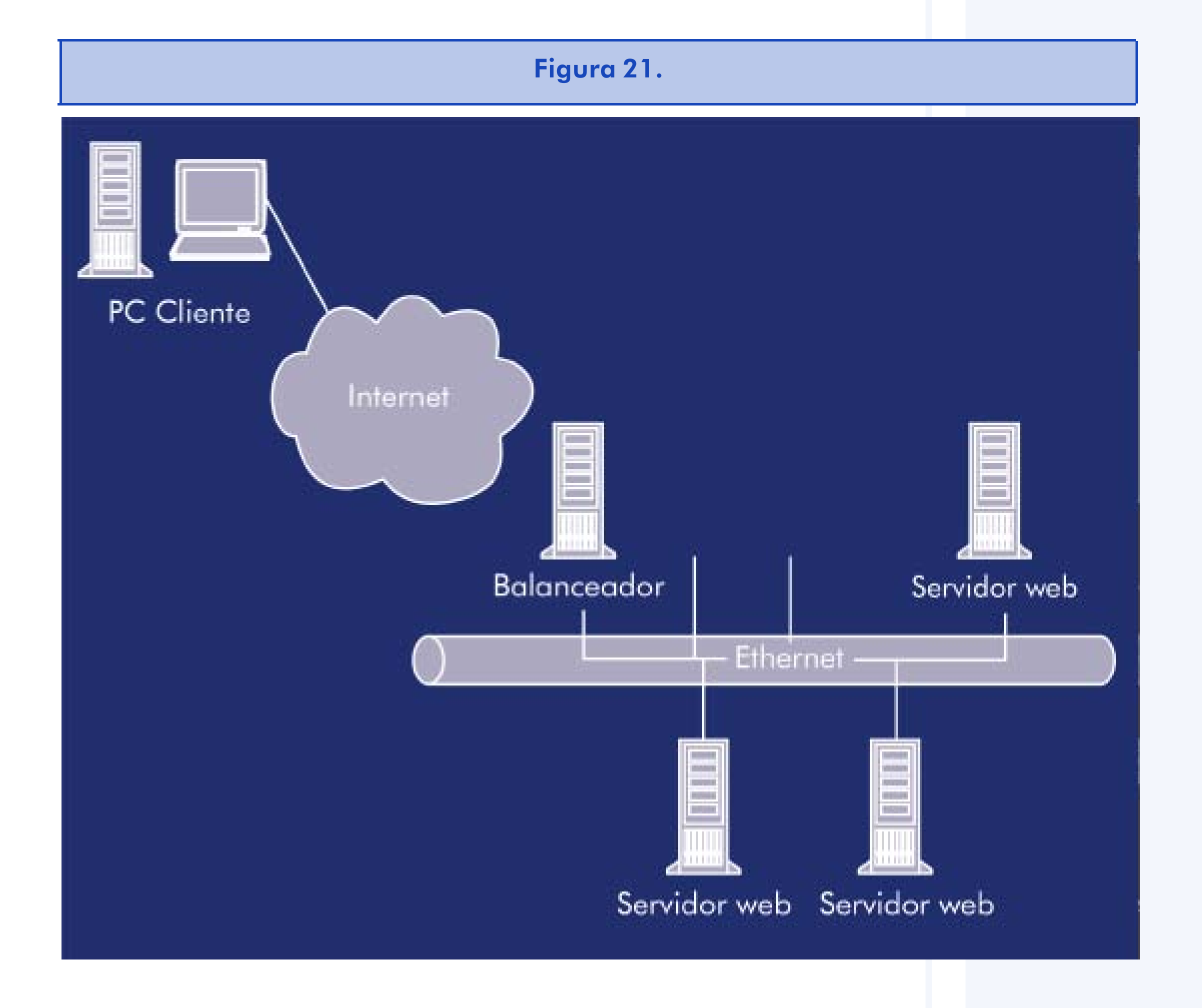
ANOTACIONES

herramientas) se basa en el proyecto LVS (Linux *Virtual Server*) y utiliza mecanismos de NAT (*Network Address Takeover*, suplantación de direcciones de red) y de IP/Port *forwarding* (reenvío de peticiones IP).

Las configuraciones basadas en LVS suelen ser configuraciones complejas y caras. Habitualmente implican a un buen número de servidores, ya que normalmente están más orientadas a proporcionar alta disponibilidad que a facilitar alto rendimiento (balanceo de carga), requiriendo, como mínimo, un sistema dedicado a balanceo de carga.

La configuración mínima de un sistema con LVS se parece a la siguiente:

Figura 21.



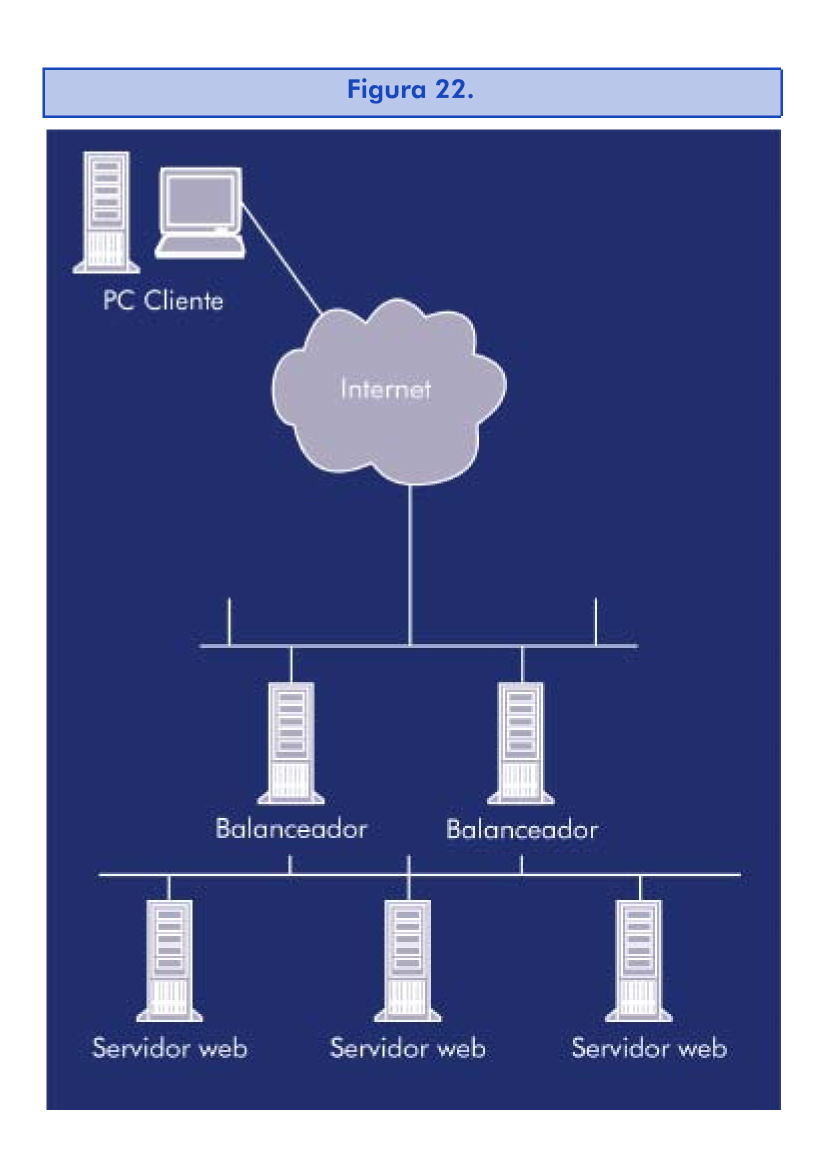
La diferencia fundamental de esta aproximación comparada con la de usar `mod_rewrite` para el balanceo de carga es que en este caso, el balanceo se realiza a nivel IP, es decir, el balanceador no tiene una copia de Apache funcionando que recibe peticiones y las reenvía, sino que realiza el reenvío a nivel IP, lo cual permite un rendimiento cientos de veces mayor.

ANOTACIONES

LVS es muy parecido (en realidad es virtualmente idéntico) a las soluciones *hardware* como Cisco, etc. La ventaja de usar LVS estriba en que al estar basado en Linux, podemos utilizar ordenadores estándar (más baratos) o incluso "reaprovechar" ordenadores, servidores, etc., obsoletos. El rendimiento de un sistema configurado para realizar tareas de balanceador es tan alto que con un ordenador tipo *Pentium III 1Ghz* podemos saturar enlaces de 100 Mbps. Esto permite reducir costos de nuestros clústers y prolongar la vida útil de los sistemas de que disponemos.

Las soluciones basadas en LVS permiten, además, utilizarse de forma dual, como soluciones de balanceo de carga para aumentar la eficiencia de nuestros servidores web y como sistemas de alta disponibilidad y resistencia a fallos. Una configuración típica de este tipo sería la siguiente:

Figura 22.



ANOTACIONES

En esta configuración los dos sistemas balanceadores se monitorizarían entre ellos para que, en caso de que se produjera una caída del que actúa como primario, el otro asumiese su tarea. Además distribuirían la carga entre los diferentes servidores web del clúster, monitorizando su correcto funcionamiento y en caso de detectar fallos, en alguno desviarían las peticiones a los restantes servidores.

Un punto que debemos tener en cuenta al implantar una solución de balanceo es la gestión de sesiones. Si tenemos, por parte de los servidores web o de algún contenedor de Servlets que estuviese dando servicio a dichos servidores web, un sistema de gestión de sesiones, conexiones a bases de datos, etc., debemos ser muy cuidadosos al implantar alguna solución de balanceo de carga. Es preciso estudiar nuestra aplicación para asegurarnos de que las peticiones de un mismo cliente a nuestra aplicación, por ejemplo, una tienda en línea, no resulten problemáticas en caso de que dichas peticiones sean atendidas alternativamente por diversos servidores web.

Muchas de estas soluciones, como LVS, proveen métodos para asegurar un mínimo de persistencia en las conexiones, es decir, que todas las peticiones procedentes del mismo cliente sean asignadas al mismo servidor durante la duración aproximada de una sesión. Pero los mecanismos ofrecidos, por el hecho de ser mecanismos de nivel de red, es decir, de nivel IP, no son infalibles.

8.2.5. Otras soluciones para el balanceo de carga

Existen otras aproximaciones para el balanceo de carga de servidores Apache que no son de carácter general, si bien pueden ser soluciones muy adecuadas para nuestro problema concreto.

OpenMOSIX o balanceo y migración de procesos

Disponemos para servidores Linux de una herramienta, llamada OpenMOSIX, que consiste en una serie de modificaciones en el núcleo del sistema operativo y que permite, en un grupo de máquinas configuradas a tal efecto, distribuir la ejecución de determinados procesos entre las diversas máquinas (a la máquina menos cargada en ese momento).

ANOTACIONES

Dicha solución no implica un balanceo real de carga, ya que los procesos sólo se “migran”, no se distribuyen y, además, cada proceso se ejecuta en una única máquina. Pero ha demostrado ser una buena solución para aquellos casos en que el problema de rendimiento de nuestra aplicación web no reside en el propio Apache, sino en el tiempo necesario para ejecutar algún CGI o programa externo. En esos casos, una solución como OpenMOSIX, que migre el proceso concreto que está tardando demasiado hacia una máquina que tenga menos carga dejando el servidor principal más descargado, puede ser muy válida. A sus ventajas se suma el hecho de que no presenta los inconvenientes propios de las otras soluciones mostrados anteriormente (control de persistencia, etc.).

Manipulación de URL o enlaces

Existe una solución, experimental y sólo disponible para Apache 1.2/1.3, que permite realizar un escalado de servicios entre máquinas muy eficiente.

Dicha solución se basa en una hipótesis que suele verificarse en la mayoría de casos y que indica que los usuarios suelen entrar a los servidores web por una serie de puntos de entrada específicos.

La solución, pues, es capaz de manipular los enlaces que aparecen en los documentos, rescribiéndolos para que apunten a alguno de los servidores del clúster. Además, puede replicar el documento enviándolo hacia éste.

Dicho módulo, llamado DC-Apache (*Distributed Cooperative* Apache *Web Server*, http://www.cs.arizona.edu/dc-apache/), permite añadir a nuestro servidor web una serie de máquinas que facilitarán la posterior derivación de la mayoría de peticiones que nos llegan hacia estas máquinas secundarias.

Para nuestro ejemplo, configuraríamos el servidor principal de la siguiente forma:

```
...
DocumentRoot "/web/www.uoc.edu/docs/"
...
   <IfModule mod_dca.c>
   SetHandler DCA-handler
```

ANOTACIONES

```
   # Directorio para recoger los documentos replicados
   ImportPath "/home/www/~migrate"

   # Directorio con los documentos que queremos
   # distribuir en el clúster
   ExportPath "/web/www.uoc.edu/docs/"

   # Servidores de apoyo
   Backend 1.2.3.2:80
   Backend 1.2.3.3:80
   Backend 1.2.3.4:80
   Backend 1.2.3.5:80
   Backend 1.2.3.6:80
   Backend 1.2.3.7:80
</IfModule>
```

Para los servidores secundarios, si no tienen ningún documento local que compartir en el clúster, la configuración sería:

```
<IfModule mod_dca.c>
   SetHandler dca-handler
   ImportPath /var/dcamigrados
   # Cuota de disco disponible para migraciones
   # p.e. 250 MB
   DiskQuota 250000000
</IfModule>
```

Distribución manual de la carga

Disponemos, obviamente, de una solución manual para el balanceo de carga, que por su simplicidad y por sus sorprendentes buenos resultados, es una de las más utilizadas. Consiste en dividir el contenido de nuestro sitio web manualmente entre varios servidores y adecuar las URL de nuestra web convenientemente.

Ejemplo

Por ejemplo, podemos poner todas las imágenes de la web en un servidor aparte del que tiene los documentos y CGI, y utilizar en todas las páginas como `IMG_SRC` lo siguiente:

```
<IMG SRC="http://imagenes.uoc.edu/logos/logo.gif">
```

ANOTACIONES

Esto desviará el tráfico requerido para los logos, etc., hacia un servidor específico, liberando el servidor principal de estas tareas.

8.3. Configuración de un *proxy-cache* con Apache

Uno de los posibles usos de Apache consiste en funcionar como servidor *proxy-cache*. El servidor puede funcionar tanto como servidor *forward-proxy* como *reverse-proxy*, y provee capacidades de *proxy* tanto para HTTP (para las versiones 0.9, 1.0 y 1.1 del protocolo) como para FTP y para CONNECT (necesario para SSL). El módulo que implementa las funcionalidades de *proxy* para Apache es muy simple, pues descarga parte de sus capacidades en otros módulos.

Ejemplo

Por ejemplo, la capacidad de almacenamiento (el *cache*) queda delegada al módulo `mod_cache`, etc.

8.3.1. Introducción al concepto de *proxy*

Un *proxy* es un servidor representante que se interpone entre un cliente que realiza una petición y el servidor que deberá resolverla. Existen multitud de situaciones en las que será aconsejable utilizar servidores *proxy* entre nuestros clientes y los servidores que deberán atender a las peticiones.

Ejemplo

Por ejemplo, se aconseja utilizar *proxy* para acelerar la navegación (caso de *forward proxy-cache*), para controlar a qué direcciones se accede (*forward proxy*), para acelerar la respuesta de un servidor web (*reverse proxy*), etc.

El *forward proxy*

Un *forward proxy* es un servidor intermediario que se sitúa entre el cliente y los servidores que deben atender a la petición.

ANOTACIONES

Para que un cliente pueda recibir el contenido de un servidor, debe realizar la petición al servidor *proxy* indicándole el servidor del que quiere obtener el contenido y la petición que quiere satisfacer. A continuación, el *proxy* realizará la petición en nombre del cliente y, una vez resuelta ésta, entregará los resultados al cliente. Generalmente en caso de *forward proxy,* los clientes deberán ser configurados específicamente para utilizar los servicios del *proxy*.

Un uso típico de *forward proxy* es el de proveer acceso a Internet a los clientes de una red interna que por motivos de seguridad se aísla de ésta, permitiendo sólo el acceso a través del *proxy*, más fácil de securizar. Otro uso muy habitual es el de realizar *cache* de páginas. En este uso el servidor *proxy* acumula localmente las páginas que van visitando los clientes. En caso de que un cliente solicite una página ya visitada, el servidor *proxy* podrá servirla directamente desde su almacenamiento local, ahorrando así ancho de banda y reduciendo el tiempo de acceso a las páginas frecuentemente visitadas (evidentemente, un *proxy-cache* deberá disponer de una política de almacenamiento/descarte de información).

Un efecto del uso de *proxy* es que para los servidores el acceso a los recursos parecerá que provenga de nuestro sistema *proxy*, no de la dirección real del cliente. Por ello es imprescindible configurar de forma muy segura los servidores *proxy* antes de conectarlos a Internet, pues podrían convertirse en un sistema de ocultación para usuarios malintencionados.

El *reverse proxy*

Un *reverse proxy*, a diferencia de un *forward proxy*, se sitúa delante de un servidor concreto, y controla el acceso a únicamente este servidor. Para los clientes el *reverse proxy* aparecerá como un servidor web normal y no se requiere ningún tipo de configuración específica.

El cliente realiza las peticiones al servidor *reverse proxy*, el cual decide dónde reenviar dichas peticiones y una vez resueltas éstas, devuelve el resultado como si el *reverse proxy* fuese la procedencia del contenido.

ANOTACIONES

Un uso típico de un *reverse proxy* consiste en filtrar y controlar el acceso de los usuarios de Internet a un servidor que deseamos esté muy aislado. Otros usos de los *reverse proxy* incluyen balanceo de carga entre servidores o proveer mecanismos de *cache* para servidores más lentos. También pueden usarse para unificar las direcciones URL de diversos servidores bajo un único espacio de nombres de URL: el del *servidor proxy*.

8.3.2. Configuración de un *forward proxy*

Para configurar un *forward proxy* debemos indicar primero a Apache que deberá usar los módulos necesarios. Éstos son:

- `mod_proxy`: el módulo que proporciona los servicios de *proxy*.
- `mod_proxy_http`: servicios para *proxy* de protocolos HTTP.
- `mod_proxy_ftp`: servicios para *proxy* de protocolo FTP.
- `mod_proxy_connect`: servicios para *proxy* de SSL.
- `mod_cache`: módulo de *cache*.
- `mod_disk_cache`: módulo de *cache* auxiliar de disco.
- `mod_mem_cache`: módulo de *cache* auxiliar de memoria.
- `mod_ssl`: módulo auxiliar de conexiones SSL.

Una vez cargados los módulos necesarios, pasaremos a configurar el servidor *proxy*. La primera configuración consistirá en indicar que Apache actuará como servidor *forward proxy*.

```
LoadModule proxy_module modules/mod_proxy.so

<IfModule mod_proxy.c>
     LoadModule http_proxy_module modules/mod_proxy_http.so
     ProxyRequests On
     ProxyVía On

     <Proxy *>
          Order deny,allow
          Deny from all
          Allow from 172.16.0.0/16
     </Proxy>
</IfModule>
```

Podemos ver en el ejemplo la simplicidad de configuración del modulo `mod_proxy`. En primer lugar activamos el manejo de peticiones de *proxy* por parte de Apache. La directiva `ProxyRequests` indica,

ANOTACIONES

en caso de estar activada, que Apache deberá actuar como un *forward proxy*. La segunda directiva `ProxyVia` indica al módulo que deberá marcar las peticiones que realice con el campo `Via`: destinada a controlar el flujo de peticiones en una cadena de *proxies*.

El bloque `Proxy` nos permite configurar las restricciones de seguridad del servidor *proxy*. En este caso sólo dejamos utilizar el servidor *proxy* a todas las máquinas pertenecientes a nuestra red interna (172.16.0.0). Podemos usar aquí todas las directivas de control de acceso de Apache.

Una vez tengamos configurado el servidor *proxy*, deberemos configurar a su vez el módulo de *cache* de Apache. Para ello definiremos un almacenamiento en disco de 256 Mbytes:

```
Sample httpd.conf
#
# Sample cache Configuration
#
LoadModule cache_module modules/mod_cache.so
<IfModule mod_cache.c>
     LoadModule disk_cache_module modules/mod_disk_cache.so
     <IfModule mod_disk_cache.c>
          CacheRoot /var/cache
          CacheSize 256
          CacheEnable disk /
          CacheDirLevels 5
          CacheDirLength 3
     </IfModule>
</IfModule>
```

8.3.3. Configuración de un *reverse proxy*

Para configurar Apache como *reverse proxy*, deberemos cargar los mismos módulos que en el caso de usarlo como *forward proxy*. Una de las diferencias estribará, eso sí, en que la configuración de las directivas de seguridad será ahora mucho menos crítica, pues indicaremos al *proxy* explícitamente a qué servidores puede acceder.

ANOTACIONES

La configuración de un *reverse proxy* para acceder a un servidor interno que tenemos en otro puerto TCP y mapearlo a un subdirectorio de nuestro espacio web sería:

```
ProxyRequests Off
     <Proxy *>
          Order deny,allow
          Allow from all
     </Proxy>
ProxyPass /interno                http://interno.uoc.edu:8181/
ProxyPassReverse /interno         http://interno.uoc.edu:8181/
```

La directiva `ProxyPass` indica a Apache que todas las peticiones destinadas a la URL indicada (`/interno`) debe convertirlas internamente en peticiones al servidor destino especificado. Además, la directiva `ProxyPassReverse` indica que las respuestas recibidas del servidor destino especificado se reescribirán como procedentes del propio servidor *proxy reverse* y provenientes del espacio URL indicado (`/interno`).

Podemos añadir a la configuración como *proxy reverse* las prestaciones de `mod_cache` para realizar almacenamiento local de las peticiones al igual que con un *forward proxy*.

8.3.4. Otras directivas de configuración

El modulo `mod_proxy` dispone, además, de otras directivas de configuración que nos permiten ajustar la operativa del servidor.

Directiva ProxyRemote/ProxyRemoteMatch

La directiva `ProxyRemote` nos permite reenviar las peticiones recibidas hacia otros servidores *proxy*, por ejemplo,

```
ProxyRemote http://www.uoc.edu/manuales/ http://manuales.uoc.edu:8000
ProxyRemote * http://servidorrapido.com
```

ANOTACIONES

Estas dos configuraciones reenviarán todas las peticiones que coincidan con `http://www.uoc.edu/manuales/` a otro servidor y todas las demás peticiones a un servidor concreto.

Disponemos de una variante de `ProxyRemote` llamada `ProxyRemoteMatch` que permite utilizar expresiones regulares para indicar la URL que vamos a comprobar.

Directiva ProxyPreserveHost

Esta directiva nos permite indicar a Apache que en las peticiones debe conservar el campo `Host` en lugar de sustituirlo por el especificado en la configuración de `ProxyPass`. Es sólo necesario en casos en que el servidor oculto tras un *reverse proxy* sea un *Virtual Host* basado en nombre.

Directiva NoProxy

Esta directiva nos permite excluir del tratamiento por `mod_proxy` aquellos ordenadores, dominios, direcciones, etc., que no deseemos.

8.4. Otros módulos de Apache

Apache dispone de multitud de módulos adicionales que podemos incluir en nuestro servidor. Algunos de estos módulos se distribuyen con el paquete oficial de Apache. Para usarlos, sólo tenemos que asegurarnos de que están presentes en nuestro servidor y activarlos en los ficheros de configuración. Otros módulos aportados por cientos de desarrolladores no se suministran de forma oficial, debiendo por ello descargarlos por separado e instalarlos en nuestro servidor.

8.4.1. mod_actions

Este módulo nos proporciona métodos para ejecutar acciones basándonos en el tipo de fichero solicitado. Una configuración simple del mismo es:

```
Action image/gif /cgi-bin/imagenes.cgi
```

ANOTACIONES

Tened en cuenta que la URL y el nombre de fichero solicitados se pasan al programa que proporciona la acción mediante las variables de entorno: CGI PATH_INFO y PATH_TRANSLATED.

8.4.2. mod_alias

Nos permite definir zonas de URL que estarán ubicadas en el disco del servidor fuera de la zona definida por `DocumentRoot`. Proporciona diversas directivas, entre ellas `Alias`, para definir nuevos directorios, `Redirect` para definir redirecciones y `ScriptAlias` para definir nuevos directorios que contendrán CGIs.

```
Alias /imagenes /web/imagenes
<Directory /web/imagenes>
     Order allow,deny
     Allow from all
</Directory>

Redirect permanent /manuales http://manuales.uoc.edu/
Redirect 303 /documento http://www.uoc.edu/enpreparacion.html.
```

8.4.3. mod_auth, mod_auth_dbm, mod_auth_digest, mod_auth_ldap

Nos permiten, junto con otros módulos no distribuidos de forma estándar, utilizar diversas fuentes de datos para la autenticación de nuestros usuarios.

8.4.4. mod_autoindex

Nos permite controlar cómo Apache generará listas de ficheros para aquellos directorios donde no exista un fichero de índice, cómo definir formatos, columnas, orden, la aparición o no de todos los ficheros, etc.

8.4.5. mod_cgi

Es el controlador principal que permite a Apache servir ficheros de tipo CGI.

ANOTACIONES

8.4.6. mod_dav y mod_dav_fs

Nos proporciona las funcionalidades de las clases 1 y 2 del estándar WebDAV, un sistema que permite manipular los contenidos de la web mucho más allá de lo especificado por el estándar HTTP. WebDAV convierte el servidor web en un virtual servidor de disco, proporcionándonos las mismas prestaciones que un servidor de disco normal: copiar, leer, mover, borrar, etc.

Nota

WebDAV es una extensión de HTTP para manipulación avanzada del contenido y facilidades de autoría.

8.4.7. mod_deflate

Permite comprimir el contenido antes de enviarlo al cliente, aumentando así la capacidad de nuestras líneas de comunicación. Debemos tener en cuenta que no todos los navegadores soportan la compresión; por ello es muy conveniente consultar la documentación de este módulo para obtener recetas muy útiles para detectar y evitar problemas con algunos navegadores. Una configuración que evita algunos problemas es la siguiente:

```
<Location />
 # Activamos filtro
 SetOutputFilter DEFLATE
 # Problema: Netscape 4.x
 BrowserMatch ^Mozilla/4 gzip-only-text/html
 # Problema: Netscape 4.06-4.08
 BrowserMatch ^Mozilla/4\.0[678] no-gzip
# MSIE
 BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
 # No comprimir imágenes
 SetEnvIfNoCase Request_URI .(?:gif jpe?g png)$ no-gzip dont-vary.
 # Evitar modificaciones por parte de proxies
 Header append Vary User-Agent env=!dont-vary
</Location>
```

8.4.8. mod_dir

Proporciona el soporte necesario para servir directorios como URL realizando los *redirects* adecuados y sirviendo los ficheros índice.

8.4.9. mod_env

Nos permite definir nuevas variables de entorno a pasar a los programas CGI. Proporciona dos directivas, `SetEnv` y `UnsetEnv`.

```
SetEnv VARIABLE_ESPECIAL valor
UnsetEnv LD_LIBRARY_PATH
```

8.4.10. mod_expires

Nos permite generar cabeceras de HTTP que indiquen la expiración de los contenidos de acuerdo a un criterio definido por nosotros:

```
# Activamos el módulo
ExpiresActive On
# expirar images GIF 1 mes después de la modificación
ExpiresByType image/gif "modification plus 1 month"
# Documentos HTML 1 semana después de la modificación
ExpiresByType text/html "modification plus 1 week"
# El resto 1 mes después del último acceso
ExpiresDefault "access plus 1 month"
```

8.4.11. mod_ldap

Proporciona un *pool* de conexiones LDAP y hace *cache* de los resultados para su uso en otros módulos que requieran LDAP. Es básico para que usar LDAP como fuente de autenticación, etc., no resulte en un cuello de botella.

8.4.12. mod_mime

Es el responsable de decidir qué tipo de fichero MIME (el estándar que marca tipos de contenido en la web) se asocia a cada fichero servido por Apache. A partir de la extensión del fichero, es capaz de decidir qué `Content-type` asociarle e incluso es capaz de decidir emprender acciones o enviar el fichero a módulos especiales para su procesamiento. Incluye un módulo `mod_mime_magic` que complementa a `mod_mime` para aquellos ficheros de los que `mod_mime` no ha podido determinar el tipo.

ANOTACIONES

8.4.13. mod_speling

Provee de un mecanismo de corrección de direcciones (URL) que los usuarios hayan introducido incorrectamente. En caso de no encontrar un recurso solicitado, Apache intentará corregir el error, por ejemplo, por un uso incorrecto de mayúsculas, etc.

En caso de encontrar más de una página candidata, mostrará al usuario la lista para que éste pueda escoger.

Hay que tener en cuenta que el impacto en el rendimiento de este módulo es sustancial.

8.4.14. mod_status

Muestra información del estado del servidor y de su nivel de ocupación y actividad. Las informaciones proporcionadas son:

- El número de procesos hijos.
- En número de procesos hijo desocupados.
- El estado de cada hijo, el número de peticiones y de bytes servidos por cada hijo.
- Número de accesos y de bytes servidos en total.
- Cuándo se inició el servidor y el tiempo que lleva funcionando.
- Medias de peticiones por segundo, bytes por segundo y bytes por petición.
- Uso actual de CPU por hijo y el total de Apache.
- Peticiones siendo procesadas actualmente.

Algunas de estas informaciones podrían estar deshabilitadas durante la compilación en nuestro servidor.

ANOTACIONES

Podemos ver cómo activar este módulo en el siguiente ejemplo:

```
<Location /estatus>
     SetHandler server-status
     Order Deny,Allow
     Deny from all
     Allow from .uoc.edu
</Location>
```

Podemos ver parte del resultado del módulo en la siguiente figura:

Figura 23.

Current Time: Tuesday, 28-Oct-2003 14:20:22 CET
Restart Time: Tuesday, 28-Oct-2003 14:17:31 CET
Parent Server Generation: 0
Server uptime: 2 minutes 51 seconds
Total accesses: 11 - Total Traffic: 36 kB
CPU Usage: u.01 s.01 cu0 cs0 - .0117% CPU load
.0643 requests/sec - 215 B/second - 3351 B/request
1 requests currently being processed, 7 idle workers

```
______W_........................................................
................................................................
................................................................
................................................................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	29375	0/1/1	_	0.01	10	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
1-0	29376	0/1/1	_	0.00	10	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
2-0	29377	0/2/2	_	0.00	10	0	0.0	0.01	0.01	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
3-0	29378	0/1/1	_	0.01	10	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
4-0	29379	0/2/2	_	0.00	9	0	0.0	0.01	0.01	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
5-0	29380	0/2/2	_	0.00	3	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /coursework HTTP/1.1
6-0	29381	0/1/1	W	0.00	0	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1
7-0	29382	0/1/1	_	0.00	11	0	0.0	0.00	0.00	80.58.51.172	bofh.udl.es	GET /server-status HTTP/1.1

8.4.15. mod_unique-id

Este módulo proporciona una variable de entorno con un identificador único para cada petición, que está garantizado que será único a un clúster de máquinas. El módulo no opera bajo Windows. La variable de entorno proporcionada es: `UNIQUE_ID`.

Para ello, utiliza un valor generado a partir de: *(ip_servidor, pid proceso, marca de tiempo, contador16*). El `contador16` es un contador de 16 bits que rota a 0 cada segundo.

8.4.16. mod_userdir

Nos permite proporcionar páginas personales a los usuarios de nuestro sistema. Este módulo presta una funcionalidad básica. Mapea un subdirectorio del directorio de trabajo de los usuarios de nuestro sistema a una URL concreta, generalmente: http://www.uoc.edu/ ˜usuario/.

Por ejemplo, la configuración más habitual:

```
UserDir public_html
```

Implicaría que las peticiones a http://www.uoc.edu/ ˜usuario/ se resolverían a partir de lo contenido en un subdirectorio `public_html` del directorio de trabajo de `usuario`.

8.4.17. mod_usertrack

Proporciona un módulo que mediante el uso de una *Cookie* realiza un seguimiento de la actividad de los usuarios a través de la web.

ANOTACIONES

9. Monitorización y análisis

9.1. Análisis de *logs* de servidores HTTP

Los servidores web (y los de FTP, *proxy-cache,* etc.) guardan, si están configurados para ello, unos ficheros en el sistema donde anotan todos los eventos que ocurren durante el funcionamiento normal del servicio. Dichos ficheros se llaman ficheros de registro o *log*. En ellos podemos encontrar el registro de

operaciones que han fallado, incluyendo algunas veces el motivo del fallo. Encontraremos también registro de operaciones anómalas, y, además, un registro de todas las operaciones realizadas correctamente.

9.1.1. Formato del fichero de *log*

Por norma general, los servidores web guardan los registros en un formato llamado *Common Log Format*. Los servidores que no usan dicho formato por defecto suelen incluir una opción para usarlo. El formato *Common Log Format* es el siguiente:

```
65.61.162.188 - - [14/Dec/2003:04:10:38 +0100] "GET /exec/rss HTTP/1.1" 200 9356
66.150.40.79 - - [14/Dec/2003:04:18:46 +0100] "HEAD / HTTP/1.1" 302 0
69.28.130.229 - - [14/Dec/2003:04:36:59 +0100] "GET /robots.txt HTTP/1.1" 404 1110
69.28.130.229 - - [14/Dec/2003:04:37:00 +0100] "GET /space/start HTTP/1.1" 200 17327
64.68.82.167 - - [14/Dec/2003:05:23:32 +0100] "GET /robots.txt HTTP/1.0" 404 1110
64.68.82.167 - - [14/Dec/2003:05:23:32 +0100] "GET / HTTP/1.0" 304 0
66.196.90.246 - - [14/Dec/2003:05:36:14 +0100] "GET /robots.txt HTTP/1.0" 404 1110
66.196.90.63 - - [14/Dec/2003:05:36:14 +0100] "GET /exec/authenticate HTTP/1.0" 302 0
66.196.90.63 - - [14/Dec/2003:05:36:19 +0100] "GET /space/start HTTP/1.0" 200 17298
69.28.130.222 - - [14/Dec/2003:05:50:32 +0100] "GET /robots.txt HTTP/1.1" 404 1110
69.28.130.222 - - [14/Dec/2003:05:50:33 +0100] "GET / HTTP/1.1" 302 14
69.28.130.222 - - [14/Dec/2003:05:50:34 +0100] "GET /space/start HTTP/1.1" 200 17327
```

Como podemos observar, cada una de las líneas del fichero de registro tiene el siguiente formato:

Tabla 20.

Nombre	Descripción
cliente remoto	Dirección IP o nombre del cliente remoto que ha realizado la petición
rfc931	Identificador de usuario remoto si éste se ha definido, – si no está definido
usuario	Identificador de usuario que se ha validado contra nuestro servidor, – si no está definido
fecha	Fecha de la petición
petición	Petición (método y URL) enviada por el cliente
estatus	Código numérico del resultado
bytes	Tamaño en bytes del resultado (0 si no procede)

El formato común extendido

Existe una variante extendida del *Common Log Format*, denominada *Extended Common Log Format* o más conocida como *Combined Log Format*, que añade al formato anterior dos campos adicionales:

```
65.61.162.188 - - [14/Dec/2003:04:10:38 +0100] “GET /exec/rss HTTP/1.1”
     200 9356 “http://www.google.com” Mozilla/4.5[en]
66.150.40.79 - - [14/Dec/2003:04:18:46 +0100] “HEAD / HTTP/1.1”
     302 0 “http://www.altavista.com” Mozilla/3.1[en]
```

Los campos adicionales que añade esta extensión son:

Tabla 21.

Nombre	Descripción
referer	La dirección de la que proviene el cliente. Si no está definida usaremos –
Agente de usuario	La versión de software del navegador que utiliza nuestro cliente. En caso de no poder determinarse, usaremos –

9.1.2. Análisis del fichero de *log*

Los ficheros de *log* nos van a proporcionar una información muy útil que nos permitirá conocer algunos datos importantes sobre los visitantes de nuestro sitio web. No obstante, muchos datos relevantes no los podremos encontrar en nuestros ficheros de *log*, por lo que de-

ANOTACIONES

beremos inferirlos de forma aproximada a partir de la información de éstos.

Los datos que vamos a poder encontrar en el fichero de *log* son:

- Número de peticiones recibidas (*hits*).
- Volumne total en bytes de datos y ficheros servidos.
- Número de peticiones por tipo de fichero (por ejemplo, HTML).
- Direcciones de clientes diferentes atendidas y peticiones para cada una de ellas.
- Número de peticiones por dominio (a partir de dirección IP).
- Número de peticiones por directorio o fichero.
- Número de peticiones por código de retorno HTTP.
- Direcciones de procedencia (*referrer*).
- Navegadores y versiones de éstos usados.

A pesar de que las informaciones que podemos obtener del análisis de los ficheros de *log* son numerosas, hay unas cuantas que no podemos obtener. De ellas, algunas resultarían de especial interés:

- Identidad de los usuarios, excepto en aquellos casos en los que el usuario se identifique por petición del servidor.
- Número de usuarios. A pesar de tener el número de direcciones IP distintas, no podemos saber de forma absoluta el número de usuarios, y más si tenemos en cuenta la existencia de servidores *proxy-cache*. Una dirección IP puede representar:
  – Un robot, araña u otro programa de navegación automático (por ejemplo, los usados por los buscadores como Google).

ANOTACIONES

– Un usuario individual con un navegador en su ordenador.

– Un servidor *proxy-cache,* que puede ser usado por cientos de usuarios.

- Datos cualitativos: motivaciones de los usuarios, reacciones al contenido, uso de los datos obtenidos, etc.

- Ficheros no vistos.

- Qué visitó el usuario al salir de nuestro servidor. Este dato quedará recogido en los *log* del servidor donde el usuario fue después del nuestro.

Hay otra información que sí queda registrada pero de forma parcial, por lo que puede llevarnos a interpretaciones erróneas de los datos. Gran parte de dichas inconsistencias proceden del *cache* que realizan los propios navegadores, del que realizan servidores *proxy-cache* intermedios, etc.

Errores comunes en la interpretación de los *logs*

La información contenida en los ficheros de *log* no permite inferir las siguientes informaciones, a pesar de que, por norma general, la mayoría de programas de análisis de *log* lo hacen:

- Los *hits* no equivalen a visitas. Una página puede generar más de un *hit,* ya que contiene imágenes, hojas de estilo, etc., que corresponden a otro *hit.*

- Las sesiones de usuario son fáciles de aislar y contar. Las sesiones, si no existe un mecanismo específico de seguimiento (*cookies,* etc.), se obtienen normalmente considerando todos los accesos provenientes de la misma dirección durante un lapso de tiempo consecutivo como perteneciente a la

- misma sesión. Esto no tiene en cuenta, ni la existencia de servidores *proxy-cache,* ni la posibilidad de que un usuario se mantenga

ANOTACIONES

un tiempo detenido (consultando otras fuentes de información, etc.).

- Datos como las medias de páginas por visita y las listas de páginas más visitadas se obtienen a partir de las sesiones de usuario. Dada la dificultad de calcular éstas, los valores obtenidos no tendrán excesiva fiabilidad. Además, la existencia de servidores *proxy-cache* tiene un efecto altamente perjudicial en las listas de páginas más visitadas. Precisamente al ser las más visitadas, tendrán más posibilidades de estar almacenadas en los servidores de *cache*.

- Es difícil deducir la ubicación geográfica de los usuarios a partir de las direcciones IP. En muchos casos, ubicaremos todo un bloque de direcciones en la ciudad donde tiene la sede principal el proveedor de servicios de Internet de un usuario, a pesar de que éste puede encontrarse en un lugar distinto.

9.1.3. Programas de análisis de *logs*

Existen múltiples programas de análisis de ficheros de *log* de código libre que podemos usar para obtener información sobre los registros de visitas de nuestro sitio web. La mayoría de ellos generan los correspondientes informes en forma de páginas web que podemos publicar incluso en nuestro sitio web.

Webalizer

Sin duda, uno de los más conocidos y usados es Webalizer. Tanto es así que incluso algunas distribuciones de Linux lo incluyen ya preconfigurado en ellas.

En caso de no disponer de Webalizer en el sistema de nuestro servidor, la configuración a partir del código fuente no presenta demasiadas dificultades.

Descargamos primero el programa del sitio web en el que se aloja y en el que podremos encontrar además documentación adicional y al-

ANOTACIONES

gunas contribuciones (`http://www.mrunix.net/webalizer`). Una vez descargado, lo descomprimimos:

```
[carlesm@bofh k]$ tar xvzf webalizer-2.01-10-src.tgz
webalizer-2.01-10/
webalizer-2.01-10/aclocal.m4
webalizer-2.01-10/CHANGES
webalizer-2.01-10/webalizer_lang.h
webalizer-2.01-10/configure
[...]
webalizer-2.01-10/sample.conf
webalizer-2.01-10/webalizer.1
webalizer-2.01-10/webalizer.c
webalizer-2.01-10/webalizer.h
webalizer-2.01-10/webalizer.LSM
webalizer-2.01-10/webalizer.png
```

Una vez descomprimido y dentro ya del directorio de construcción, podemos configurar su compilación:

```
[carlesm@bofh webalizer-2.01-10]$ ./configure \
     --with-language=spanish --prefix=/home/carlesm/web
creating cache ./config.cache
checking for gcc... gcc
[...]
creating Makefile
linking ./lang/webalizer_lang.spanish to webalizer_lang.h
[carlesm@bofh webalizer-2.01-10]$
```

Disponemos de una opción importante, `with-language`, que nos permitirá especificar el idioma con el que queremos construir e instalar Webalizer. Para escoger el idioma, miramos dentro del subdirectorio `lang` para ver los idiomas disponibles.

Construimos ahora el programa como es habitual en estos casos con el comando `make`.

```
[carlesm@bofh webalizer-2.01-10]$ make
gcc -Wall -O2 -DETCDIR=\"/etc\" -DHAVE_GETOPT_H=1
-DHAVE_MATH_H=1 -c webalizer.c
[...]
gcc -o webalizer webalizer.o hashtab.o linklist.o preserve.o
  parser.o output.o dns_resolv.o graphs.o -lgd -lpng -lz -lm
rm -f webazolver
ln -s webalizer webazolver
[carlesm@bofh webalizer-2.01-10]$
```

ANOTACIONES

Y, una vez construido, lo instalamos:

```
[carlesm@bofh webalizer-2.01-10]$ make install
/usr/bin/install -c webalizer /home/carlesm/web/bin/webalizer
[...]
rm -f /home/carlesm/web/bin/webazolver
ln -s /home/carlesm/web/bin/webalizer \
    /home/carlesm/web/bin/webazolver
[carlesm@bofh webalizer-2.01-10]$
```

Para generar un informe de *logs,* podemos ejecutarlo pasándole como parámetro un fichero de *logs* y, en el directorio actual, nos dejará los ficheros que contendrán dicho informe.

```
[carlesm@bofh log]$ ~/web/bin/webalizer access_log
Webalizer V2.01-10 (Linux 2.4.20-8) Spanish
Utilizando histórico access_log (clf)
Creando informe en directorio actual
El nombre de máquina en el informe es 'bofh'
No encuentro el archivo histórico...
Generando informe de Diciembre 2003
Generando informe resumido
Guardando información de archivo...
45 registros en 0.03 segundos
[carlesm@bofh log]$ ls
access_log
ctry_usage_200312.png
daily_usage_200312.png
hourly_usage_200312.png
index.html
usage_200312.html
usage.png
webalizer.hist
[carlesm@bofh log]$
```

Podemos obtener así un informe de utilización a partir del *log* de nuestro servidor web:

ANOTACIONES



Figura 24.



Estadísticas de uso para bofh

Periodo resumido: Últimos 12 meses
Generado el 12-Dec-2003 23:14 CET

Resumen por meses										
Mes	Media diaria				Totales mensuales					
	Accesos	Archivos	Páginas	Visitas	Clientes	KBytes	Visitas	Páginas	Archivos	Accesos
Dic 2003	9	0	4	3	30	46	17	20	3	45
Totales						**46**	**17**	**20**	**3**	**45**

Generated by Webalizer Version 2.01

Figura 25.

[Estadísticas diarias] [Estadísticas por horas] [URLs] [Entrada] [Salida] [Clientes] [Enlaces origen] [Búsqueda] [Navegadores] [Países]

Estadísticas mensuales de Diciembre 2003		
Total Accesos		45
Total Archivos		3
Total Páginas		20
Total Visitas		17
Total KBytes		46
Total Clientes		30
Total URLs		2
Total Enlaces origen		1
Total Navegadores		6
	Media	**Max**
Accesos por Hora	0	6
Accesos por Día	9	13
Archivos por Día	0	1
Páginas por Día	4	5
Visitas por Día	3	5
KBytes por Día	9	14
Accesos por código de respuesta		
200 - OK		3
400 - Petición errónea		1
403 - Prohibido		2
404 - No se encuentra		30
405 - Método no permitido		8
413 - Entidad de petición demasiado grande		1

ANOTACIONES

Awstats

AWstats es un programa de estadísticas y análisis de *logs* que tiene una lista de prestaciones y capacidades muy completa. AWstats puede generar, además de las estadísticas de servidores web, estadísticas de servidores de correo, ficheros, etc.

Este analizador de *logs* puede funcionar tanto como un módulo CGI como desde la línea de comandos. Algunas de las informaciones que nos ofrece son:

- Número de visitas y de visitantes.
- Duración de las visitas.
- Usuarios autenticados y visitas autenticadas.
- Horas y días de más tráfico (páginas, *hits*, bytes, etc.).
- Dominios y países de procedencia de las visitas.
- Páginas más visitadas y páginas de entrada/salida.
- Tipos de ficheros solicitados.
- Robots visitantes (procedentes de buscadores, etc.).
- Buscadores de procedencia de los visitantes, incluyendo las palabras y frases usadas para la búsqueda.
- Errores y códigos de retorno de HTTP.
- Características de los navegadores (Java, Flash, etc.) y tamaño de la pantalla.
- Estadísticas de compresión (si se usa ésta).
- Navegadores usados para visitarnos (versiones de navegador, páginas servidas para cada navegador, bytes por navegador, etc.).
- Sistemas operativos usados para visitarnos.

ANOTACIONES

Debemos recordar que todos estos informes y estadísticas se obtienen a partir de los datos del fichero de *log*, de los que ya hemos comentado los problemas debidos al uso de *proxy-cache*s, etc.

Para instalar el programa, siempre que nuestro sistema no lo traiga instalado de serie, debemos asegurarnos de que disponemos, como mínimo, de un intérprete de Perl. Empezaremos la instalación descargándonos del servidor web del programa el código de éste (`http://awstat.sf.net`).

- Una vez descargado los descomprimimos, como es habitual, con el comando `tar`.

```
[carlesm@bofh aws]$ tar xvzf awstats-5.9.tgz
awstats-5.9/
awstats-5.9/docs/
awstats-5.9/docs/awstats.htm
[....]
awstats-5.9/wwwroot/icon/other/vv.png
awstats-5.9/wwwroot/js/
awstats-5.9/wwwroot/js/awstats_misc_tracker.js
[carlesm@bofh aws]$
```

- El primer paso de la instalación consiste en configurar el servidor web para que utilice el formato de *log* llamado *NCSA combined/XLF/ELF*. En Apache, por ejemplo, en el fichero de configuración *httpd.conf*, cambiaremos para ello:

```
CustomLog /var/log/httpd/access_log common
```

por

```
CustomLog /var/log/httpd/access_log combined
```

- El siguiente paso es copiar el contenido del subdirectorio `wwwroot/cgi-bin/`, incluyendo los subdirectorios que contiene, en el directorio donde nuestro servidor web encontrará los ficheros CGI.

- Ahora debemos copiar el contenido de `wwwroot/icon` en un subdirectorio del directorio donde almacenemos el contenido de nuestro servidor web.

ANOTACIONES

- Debemos copiar el fichero `awstats.model.conf` en otro fichero que llamaremos `awstats.nombreservidor.conf` y ubicarlo en uno de los siguientes directorios: `/etc/awstats`, `/etc/opt/awstats`, `/etc` o en el directorio donde se encuentre `awstats.pl`.

- Configuraremos dicho fichero. Debemos, como mínimo, editar las siguientes variables:

  – **LogFile**: ruta hacia el fichero de *log*.

  – **LogType**: tipo de fichero de *log*: **W** para web, **M** para correo, **F** para FTP y **O** en otro caso.

  – **LogFormat**: debemos asegurarnos que tenga 1.

  – **DirIcons**: ruta hacia el directorio donde están los iconos.

  – **SiteDomain**: nombre del servidor web.

- Una vez configurado el programa realizaremos la primera ejecución de AWStats desde la línea de comandos, desde el directorio `cgi-bin`, con:

```
awstats.pl -config=nombreservidor -update
```

- Podemos ahora acceder a visitar nuestras estadísticas con la URL siguiente:

```
http://nombreservidor/cgi-bin/awstats.pl
```

  Esto nos mostrará las estadísticas generadas dinámicamente. Otra opción consiste en generar las estadísticas como páginas HTML estáticas y acceder a ellas. Para ello ejecutamos:

```
awstats.pl -config=nombreservidor -output -staticlinks >
awstats.servidor.html
```

ANOTACIONES

Movemos el fichero generado (`awstats.servidor.html`) a un directorio accesible por el servidor web y ya podemos acceder a él desde un navegador.

- Debemos, a pesar de que podamos usar la actualización dinámica desde el navegador, programar el servidor para que realice una actualización periódica de las estadísticas con el comando usado anteriormente para crearlas. Para ello podemos usar las facilidades de `cron` en nuestros sistemas Linux y complementarlas, si disponemos de él, de `logrotate` para que en el momento de cambio de ficheros `log` (que se realiza para ahorrar espacio en disco) incorpore éstos a las estadísticas.

Podemos ver ahora parte del aspecto que tienen las estadísticas generadas:

Figura 26.

Hours

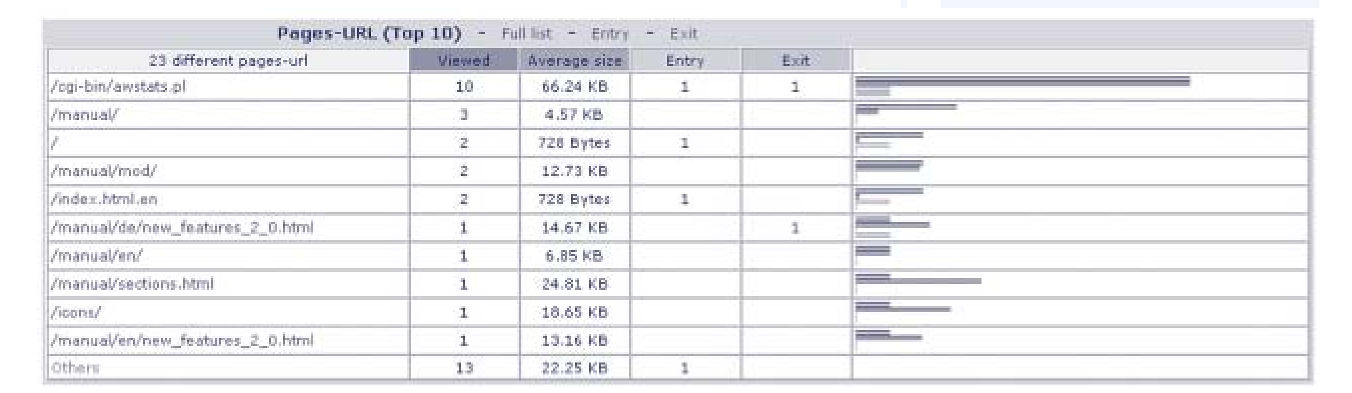
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Hours	Pages	Hits	Bandwidth	Hours	Pages	Hits	Bandwidth
00	0	0	0	12	0	0	0
01	0	0	0	13	0	0	0
02	0	0	0	14	0	0	0
03	0	0	0	15	0	0	0
04	0	0	0	16	0	0	0
05	0	0	0	17	0	0	0
06	0	0	0	18	0	0	0
07	0	0	0	19	7	23	142.58 KB
08	0	0	0	20	8	11	346.93 KB
09	0	0	0	21	0	0	0
10	0	0	0	22	0	0	0
11	0	0	0	23	0	0	0

Visitors domains/countries (Top 10) - Full list

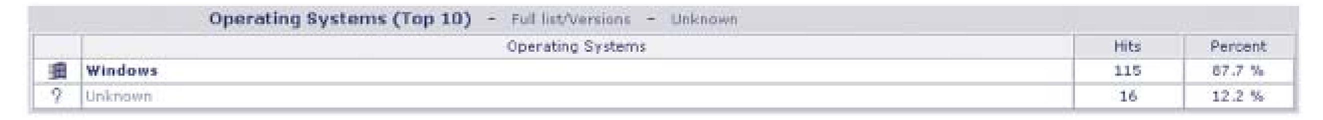
	Domains/Countries		Pages	Hits	Bandwidth
?	Unknown	ip	15	34	489.50 KB

Figura 27.



Pages-URL (Top 10) - Full list - Entry - Exit

23 different pages-url	Viewed	Average size	Entry	Exit
/cgi-bin/awstats.pl	10	66.24 KB	1	1
/manual/	3	4.57 KB		
/	2	728 Bytes	1	
/manual/mod/	2	12.73 KB		
/index.html.en	2	728 Bytes	1	
/manual/de/new_features_2_0.html	1	14.67 KB		1
/manual/en/	1	6.85 KB		
/manual/sections.html	1	24.81 KB		
/icons/	1	18.65 KB		
/manual/en/new_features_2_0.html	1	13.16 KB		
Others	13	22.25 KB	1	



Operating Systems (Top 10) - Full list/Versions - Unknown

	Operating Systems	Hits	Percent
	Windows	115	87.7 %
?	Unknown	16	12.2 %

Browsers (Top 10) - Full list/Versions - Unknown

	Browsers	Grabber	Hits	Percent
	Firebird	No	70	53.4 %
	Mozilla	No	45	34.3 %
	Lynx	No	16	12.2 %

Figura 28.

Visitors domains/countries (Top 10) - Full list

	Domains/Countries		Pages	Hits	Bandwidth
?	Unknown	ip	37	131	1.14 MB

Hosts (Top 10) - Full list - Last visit - Unresolved IP Address

Hosts : 0 Known, 2 Unknown (unresolved ip) - 2 Unique visitors	Pages	Hits	Bandwidth	Last visit
10.0.0.2	21	115	902.09 KB	24 Dec 2003 - 17:02
10.69.1.3	16	16	262.98 KB	24 Dec 2003 - 17:12

Robots/Spiders visitors (Top 10) - Full list - Last visit

0 different robots	Hits	Bandwidth	Last visit

Visits duration

Number of visits: 4 - Average: 307 s	Number of visits	Percent
0s-30s	1	25 %
30s-2mn		
2mn-5mn		
5mn-15mn	1	25 %
15mn-30mn		
30mn-1h		
1h+		
Unknown	2	50 %

Analog

Analog es quizás el más antiguo y usado de los programas de análisis de *logs* de código libre. Suele usarse conjuntamente con otro

programa, ReportMagic, que complementa las funcionalidades de visualización de informes de Analog.

Para instalarlo, nos dirigimos a la página web del programa, en la dirección http://www.analog.cx. Allí encontraremos versiones precompiladas para la mayoría de plataformas y versiones con código fuente. Descargamos la versión de código fuente y procederemos a su instalación:

- Descomprimimos el código del programa:

```
[carlesm@bofh l]$ tar vxzf analog-5.91beta1.tar.gz
analog-5.91beta1/
analog-5.91beta1/docs/
analog-5.91beta1/docs/LicBSD.txt
[...]
analog-5.91beta1/anlgform.pl
analog-5.91beta1/logfile.log
[carlesm@bofh l]$
```

- Entramos ahora en el directorio `src` y editamos el fichero `anlghead.h`. En este fichero se definen algunas variables de configuración: el nombre del servidor, etc. Cambiamos los valores a aquellos que deseemos.

- Compilamos con `make`:

```
[carlesm@bofh src]$ make
gcc -O2 -DUNIX -c alias.c
gcc -O2 -DUNIX -c analog.c
[...]
bzip2/huffman.o bzip2/randtable.o -lm
***
***IMPORTANT: You must read the licence before using analog
***
[carlesm@bofh src]$
```

- Podemos editar el fichero `analog.cfg` para definir el formato de salida de Analog, así como algunos de los parámetros de funcionamiento de éste.

- Una vez editado, ejecutamos Analog con `analog` para que nos genere el fichero de estadísticas.

ANOTACIONES

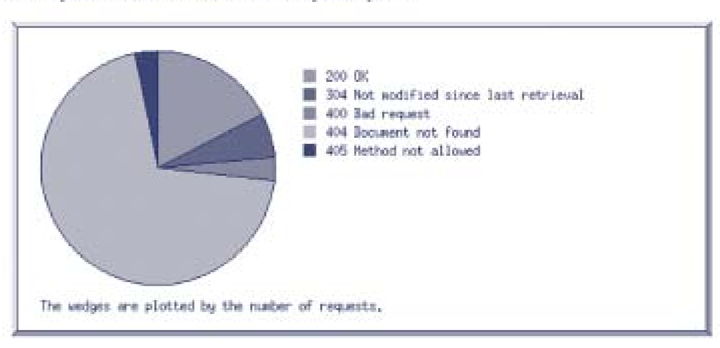
El aspecto que ofrecen las estadísticas una vez generadas es el siguiente:

Figura 29.

Status Code Report

(**Go To**: Top | General Summary | Monthly Report | Daily Summary | Hourly Summary | Domain Report | Organisation Report | Operating System Report | **Status Code Report** | File Size Report | File Type Report | Directory Report | Request Report)

This report lists the HTTP status codes of all requests.



Listing status codes, sorted numerically.

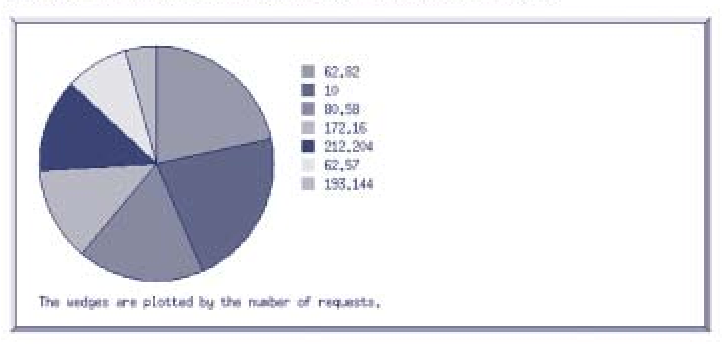
reqs	status code
17	200 OK
6	304 Not modified since last retrieval
3	400 Bad request
68	404 Document not found
3	405 Method not allowed

Figura 30.

Organisation Report

(**Go To**: Top | General Summary | Monthly Report | Daily Summary | Hourly Summary | Domain Report | **Organisation Report** | Operating System Report | Status Code Report | File Size Report | File Type Report | Directory Report | Request Report)

This report lists the organisations of the computers which requested files.



Listing organisations, sorted by the number of requests.

reqs	%bytes	organisation
5	17.65%	62.82
5	29.41%	10
4		80.58
3	17.65%	172.16
3	17.65%	212.204
2	11.76%	62.57
1	5.88%	193.144

Figura 31.

General Summary

(Go To: Top | General Summary | Monthly Report | Daily Summary | Hourly Summary | Domain Report | Organisation Report | Operating System Report | Status Code Report | File Size Report | File Type Report | Directory Report | Request Report)

This report contains overall statistics.

Successful requests: 23
Average successful requests per day: 3
Successful requests for pages: 23
Average successful requests for pages per day: 3
Failed requests: 74
Distinct files requested: 1
Distinct hosts served: 16
Data transferred: 4.67 kilobytes
Average data transferred per day: 816 bytes

Monthly Report

(Go To: Top | General Summary | Monthly Report | Daily Summary | Hourly Summary | Domain Report | Organisation Report | Operating System Report | Status Code Report | File Size Report | File Type Report | Directory Report | Request Report)

This report lists the activity in each month.

Each unit (+) represents 1 request for a page.

month	reqs	pages	
Dec 2003	23	23	+++++++++++++++++++++++

Busiest month: Dec 2003 (23 requests for pages).

9.2. Herramientas de estadísticas y contadores

9.2.1. Contadores

Los contadores web son indicaciones visuales al visitante de nuestra página de cuántas visitas hemos tenido. Constituyen un indicador visual con más valor estético que útil, ya que muchos de estos contadores no tienen ningún valor desde el punto de vista estadístico, porque sólo cuentan *hits* (peticiones de la página al servidor, que no siempre corresponden a visitas reales).

Para poner un contador en nuestra página disponemos de diversas opciones:

- Utilizar un CGI o Servlet que nos genere el contador (mediante las imágenes o las referencias a las imágenes).

- Utilizar un servicio de contador que nos muestre el contador o las referencias a las imágenes.

ANOTACIONES

- Utilizar, en el caso de que nos lo proporcione, una extensión del servidor para el contador.

Uso de un CGI para generar el contador

Para añadir un contador de visitas a nuestra página web podemos usar un programa externo, sea un CGI o un Servlet, que nos lleve la cuenta de visitas y nos genere el contador. Usaremos uno de los múltiples contadores existentes, `Count`.

Para empezar debemos descargarnos `Count` de la página web correspondiente:

```
http://www.muquit.com/muquit/software/Count/Count.html
```

Una vez descargado, empezaremos el proceso de instalación.

- El primer paso será descomprimir el código del programa. Para ello recurrimos una vez más a `tar`:

```
[carlesm@bofh n]$ tar xvzf wwwcount2.6.tar.gz
./wwwcount2.6/
./wwwcount2.6/DIR/
[...]
./wwwcount2.6/utils/rgbtxt2db/rgb.txt
./wwwcount2.6/utils/rgbtxt2db/rgbtxt2db.c
[carlesm@bofh n]$
```

- Compilamos ahora el programa:

```
[carlesm@bofh wwwcount2.6]$ ./build \
   --with-cgi-bin-dir=/home/carlesm/apache/cgi-bin/ \
   --prefix=/home/carlesm/apache/counter
```

  El parámetro `prefix` indica dónde queremos que Counter guarde sus ficheros.

- Una vez compilado lo instalamos con:

```
[carlesm@bofh wwwcount2.6]$ ./build --install
```

ANOTACIONES

El programa nos permitirá confirmar los directorios de instalación antes de copiar allí los ficheros.

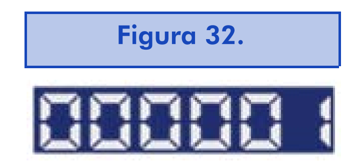
- Debemos configurar Counter. Para ello editamos el fichero `count.cfg` que se encuentra en el directorio `conf` de la instalación de Counter, en nuestro caso, de: `/home/carlesm/apache/counter`.

- Podemos poner el siguiente retazo de HTML que referencia nuestro CGI de contador para usar Counter:

```
<img src="/cgi-bin/Count.cgi" alt="Contador">
```

El aspecto de dicho contador es:

Figura 32.



- Podemos usar los diversos parámetros de Counter para modificar qué y cómo se muestra:

```
Visitas:
<img src="/cgi-bin/Count.cgi?dd=C&ft=6&frgb=ff0000" alt="******">
<br>
Hora:
<img src="/cgi-bin/Count.cgi?display=clock&pad=F" alt="Count">
<br>
Tiempo desde 1/1/2000:
<img
src="/cgi-bin/Count.cgi?cdt=2000;1;1;0;0;0&dd=cd&ft=2&frgb=000000"
alt="Count">
<br>
```

Podemos ver ahora en la imagen el aspecto que ofrecen estos tres contadores:

Figura 33.



Servicios de contadores

Existen diversos servicios de contadores proporcionados comercialmente, aunque muchos de ellos ofrecen modalidades gratuitas, que nos permiten poner un contador en nuestra página web sin necesidad de instalar ningún programa adicional en nuestro servidor. Muchos de estos contadores nos ofrecen, además, análisis estadísticos de las visitas.

Algunos de estos servicios son:

Tabla 22. Servicios de contadores

Nombre	Dirección
123 Counter	http://www.123counter.com/
Admo Free Counters	http://www.admo.net/counter/
BeSeen: Hit Counter	http://www.beseen.com/hitcounter/
BoingDragon: Animated Counters	http://www.boingdragon.com/types.html
Dark Counter	http://www.lunamorena.net/counter/
Digits.com	http://www.digits.com/create.html
Easy Counter	http://www.easycounter.com/
i-Depth	http://www.i-depth.com/X/guru3#hcnt
LBInet Counters	http://www.lbi.net/c50000/
LunaFly: Free Counter	http://www.freecount.co.uk/
MyComputer Counter	http://counter.mycomputer.com/
Spirit Counters	http://www.thesitefights.com/userv/

ANOTACIONES

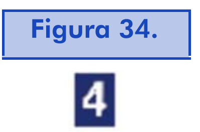
Crearemos un contador. Para ello usaremos el servicio que proporciona Digits.com. Nos dirigimos primero a su página web y una vez allí, rellenamos el formulario de petición de servicio.

Una vez rellenado el formulario, Digits.com nos proporciona un retazo de código HTML que debemos incluir en nuestra página. Dicho código es parecido al siguiente:

```
<IMG SRC="http://counter.digits.com/wc/-d/4/carlesm"
   ALIGN=middle
WIDTH=60 HEIGHT=20 BORDER=0 HSPACE=4 VSPACE=2>
```

El aspecto de dicho contador es:

Figura 34.



Podemos, mediante los parámetros pasados a la URL, cambiar el aspecto del contador, para así obtener:

Figura 35.



mediante el siguiente código HTML:

```
<p>
Visitas:
<IMG
      SRC="http://counter.digits.com/wc/-d/4/-z/-c/8/carlesm"
      ALIGN=middle
      WIDTH=60 HEIGHT=20 BORDER=0 HSPACE=4 VSPACE=2>
</p>
<p>
Visitas:
<IMG
      SRC="http://counter.digits.com/wc/-d/4/-z/-c/26/carlesm"
      ALIGN=middle
      WIDTH=60 HEIGHT=20 BORDER=0 HSPACE=4 VSPACE=2>
</p>
```

Extensión del servidor (Roxen)

El servidor web de código libre Roxen proporciona una extensión del lenguaje HTML que nos permite implementar un contador de forma sencilla sin que sea necesaria la instalación de ningún software adicional a nuestro sistema.

Para ello disponemos de dos etiquetas de HTML nuevas: `accessed` y `gtext`, que sirven para indicar el número de *hits* a una página y para mostrar un texto en forma gráfica, respectivamente.

Un ejemplo de la utilización de dicha extensión es el siguiente código:

```
Visitas:
  <gtext 2 bshadow=1 bevel=2 ><accessed /></gtext><br />
Visitas:
<b><accessed  /></b>
```

Consiguiendo con ello el resultado siguiente:

Figura 36.

Visitas: 15
Visitas: 15

9.2.2. Estadísticas de visitas

Otra de las opciones que tenemos para controlar el número de visitantes de nuestras páginas web, la procedencia de éstos y otros datos similares, sin que nos suponga usar un programa de análisis de *logs*, es hacer uso de alguno de los servicios de estadísticas y recuento de visitantes que se ofrecen, algunos de forma gratuita, en Internet.

Nota

La etiqueta `<counter>`. Por compatibilidad con versiones anteriores, donde disponíamos de una etiqueta concreta para realizar contadores, `<counter>`, Roxen nos proporciona aún dicha etiqueta implementada ahora como una combinación de `accessed` y `gtext`.

ANOTACIONES

Algunos de estos servicios los tenemos detallados en la lista siguiente:

Tabla 23. Servicios recuento visitantes

Nombre	Dirección
Counted	http://www.counted.com
Cyber Stats	http://www.pagetools.com/cyberstats/
Gold Stats	http://www.goldstats.com
Hit Box	http://www.websidestory.com
IPSTAT II	http://www.ipstat.com
NedStat	http://www.nedstat.com
RealTracker	http://www.showstat.com
Site-Stats	http://www.site-stats.com
Site Tracker	http://www.sitetracker.com
Stats 3D	http://www.stats3d.com
Stat Trax	http://www.stattrax.com
The-Counter.net	http://www.the-counter.net
WebStat.com	http://www.webstat.com
WebTrends Live	http://www.webtrendslive.com/default.htm
WhozOnTop	http://world.icdirect.com/icdirect/hitTracker.asp

El funcionamiento de la mayoría de estos servicios es muy similar. Al registrarnos (sea en la modalidad gratuita o en la de pago), nos ofrecerán un código HTML que debemos incluir en nuestras páginas. Este código de forma general referencia una imagen proveniente del sitio web del servicio de estadísticas. Algunos de estos servicios ofrecen una imagen que sirve de contador.

Un ejemplo de dicho código, correspondiente al servicio NedStat, es:

```
<!-- Begin Nedstat Basic code -->
<!-- Title: Carlesm Homepage -->
<!-- URL: http://carlesm/ -->
<script language="JavaScript"
     src="http://m1.nedstatbasic.net/basic.js">
</script>
<script language="JavaScript">
<!--
nedstatbasic("AA7hmw77L/vVx928ONUhsGLjd6mQ", 0);
// -->
</script>
```

ANOTACIONES

```
<noscript>
<a target=_blank
      href=“http://v1.nedstatbasic.net/stats?AA7hmw77L/vVx928ONUhsGLjd6mQ”>
<img
      src=“http://m1.nedstatbasic.net/n?id=AA7hmw77L/vVx928ONUhsGLjd6mQ”
      border=0 nosave width=18 height=18>
</a>
</noscript>
<!-- End Nedstat Basic code -->
```

Una vez incluido dicho código dentro de nuestra página, el servicio de estadísticas controlará cada vez que se acceda a nuestra página. Podremos acceder entonces a la visualización de las estadísticas correspondientes a nuestra página, como en los siguientes ejemplos:

Figura 37.

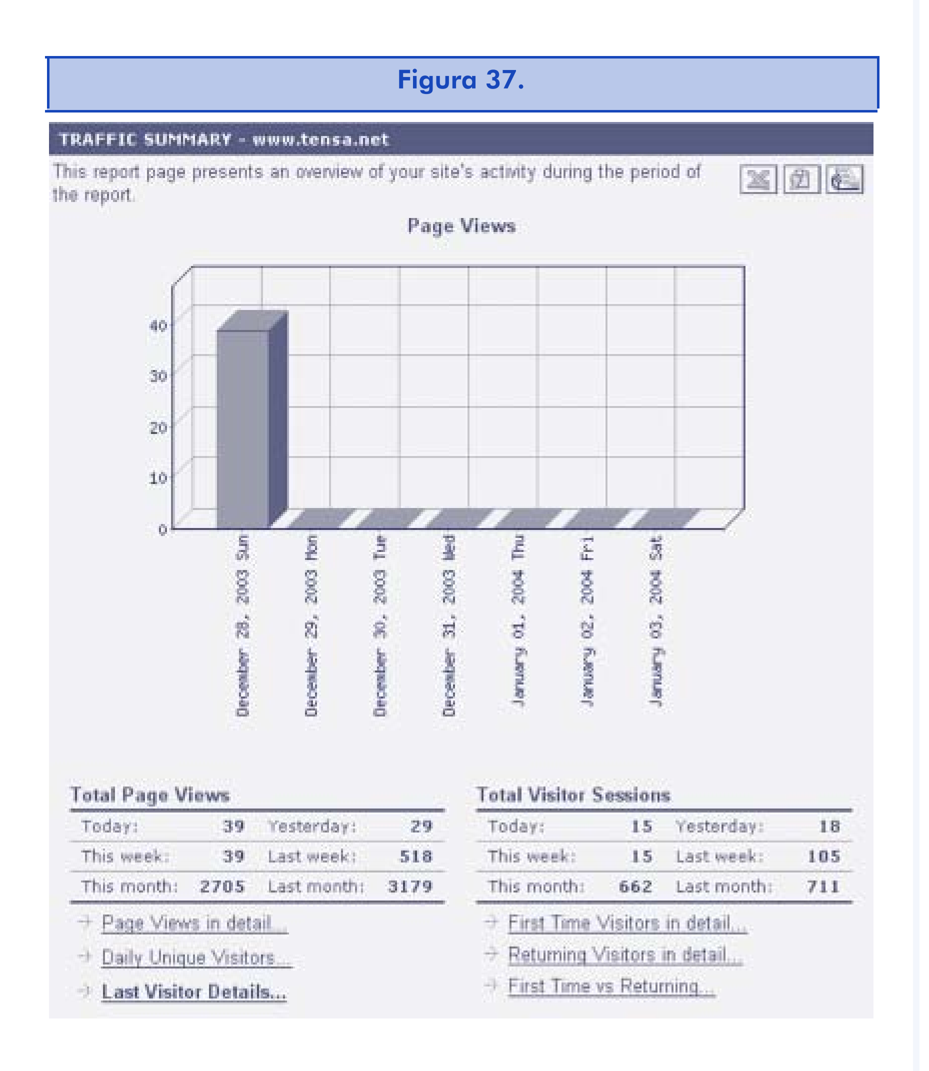
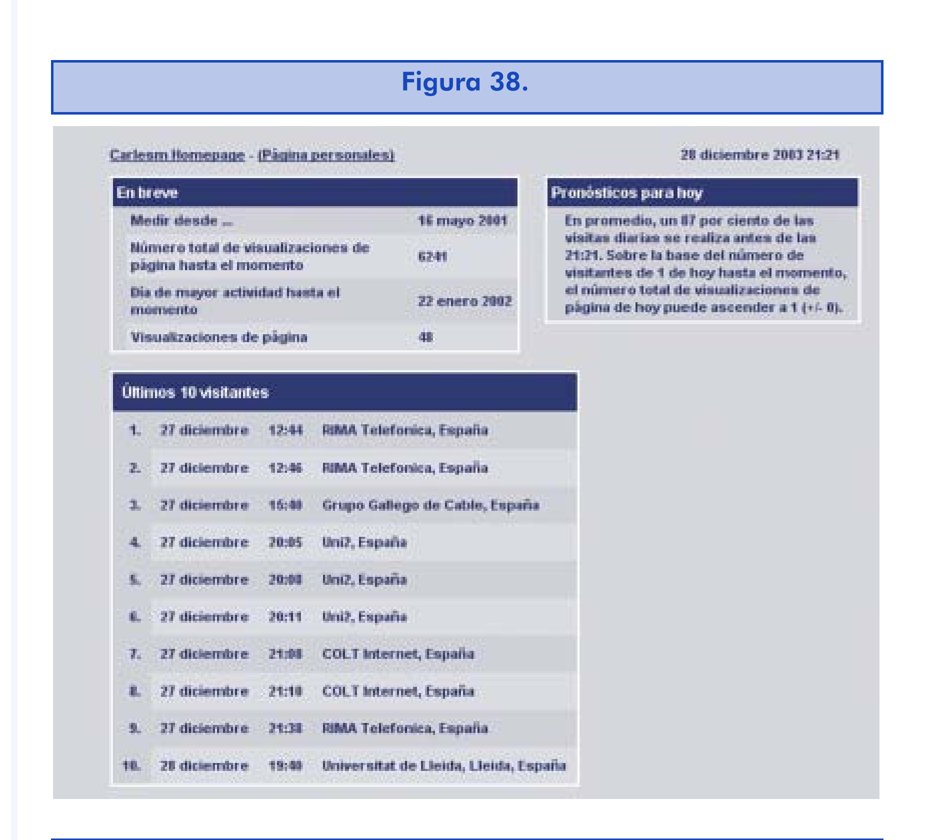


Figura 38.



Carlesm Homepage - (Página personales) 28 diciembre 2003 21:21

En breve	
Medir desde ...	16 mayo 2001
Número total de visualizaciones de página hasta el momento	6241
Día de mayor actividad hasta el momento	22 enero 2002
Visualizaciones de página	48

Pronósticos para hoy

En promedio, un 87 por ciento de las visitas diarias se realiza antes de las 21:21. Sobre la base del número de visitantes de 1 de hoy hasta el momento, el número total de visualizaciones de página de hoy puede ascender a 1 (+/- 0).

Últimos 10 visitantes			
1.	27 diciembre	12:44	RIMA Telefonica, España
2.	27 diciembre	12:46	RIMA Telefonica, España
3.	27 diciembre	15:40	Grupo Gallego de Cable, España
4.	27 diciembre	20:05	Uni2, España
5.	27 diciembre	20:08	Uni2, España
6.	27 diciembre	20:11	Uni2, España
7.	27 diciembre	21:08	COLT Internet, España
8.	27 diciembre	21:10	COLT Internet, España
9.	27 diciembre	21:38	RIMA Telefonica, España
10.	28 diciembre	19:40	Universitat de Lleida, Lleida, España

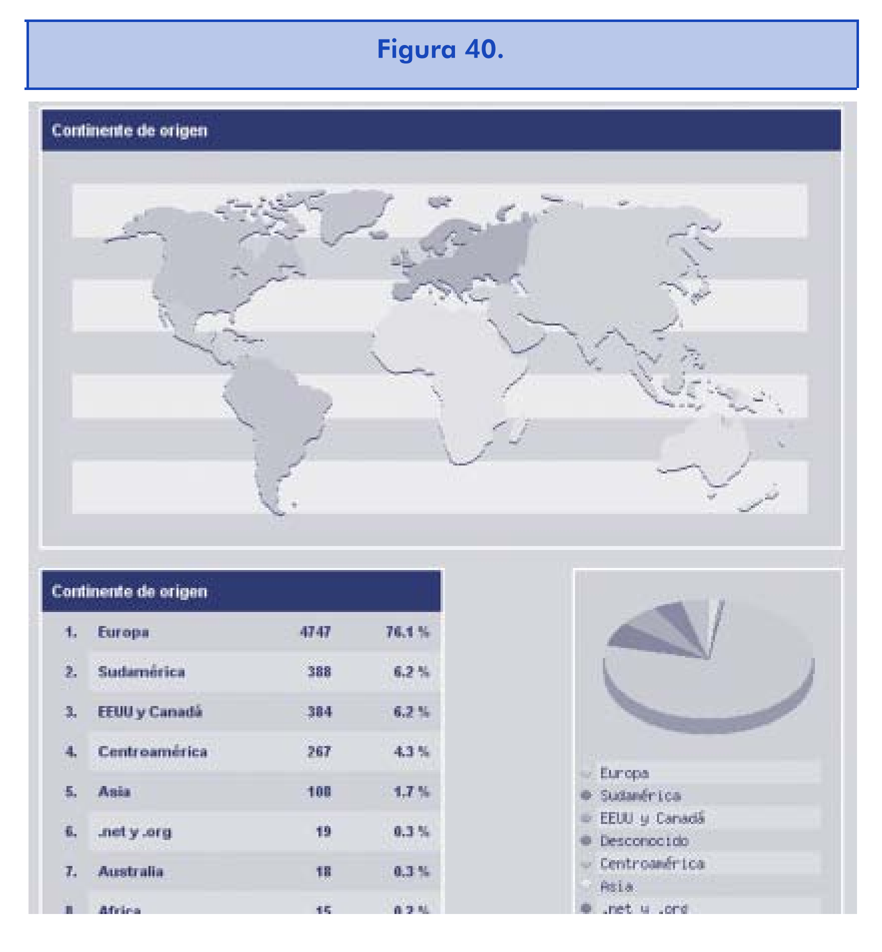


Figura 39.



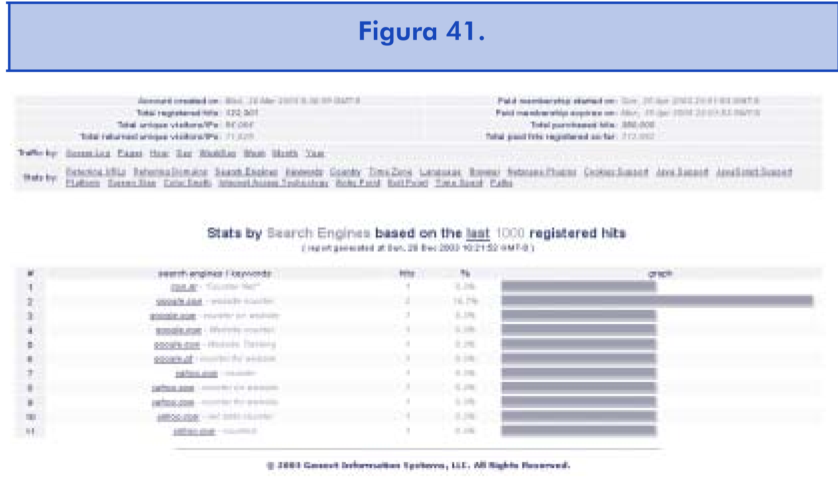
ANOTACIONES

Figura 40.



1.	Europa	4747	76.1 %
2.	Sudamérica	388	6.2 %
3.	EEUU y Canadá	384	6.2 %
4.	Centroamérica	267	4.3 %
5.	Asia	108	1.7 %
6.	.net y .org	19	0.3 %
7.	Australia	18	0.3 %
8.	Africa	15	0.2 %

Figura 41.



9.3. Análisis de rendimiento

Uno de los puntos clave del éxito de un sitio web será el nivel de comodidad de nuestros usuarios, que la experiencia al visitar nuestro sitio sea agradable, que la respuesta que obtengan a sus acciones sea fluida, sin retrasos en las respuestas, etc. Otro de estos puntos clave

será el rendimiento que obtengamos de nuestros sistemas. A mayor rendimiento, mejor aprovechamiento de la inversión. En muchos casos, ello también se traducirá en una respuesta más agradable a nuestros usuarios, más fluida, con tiempos de acceso menores, etc.

9.3.1. Obtener información de rendimiento de Apache

El primer punto que nos ofrecerá información sobre cómo está funcionando el servidor web es el propio servidor web.

mod_status

Como ya hemos visto, Apache dispone de un módulo, llamado `mod_status`, que nos muestra una página de información sobre el rendimiento en ese momento del servidor web. Recordemos el aspecto que ofrecía dicha página:

Figura 42.

```
Current Time: Tuesday, 28-Oct-2003 14:20:22 CET
Restart Time: Tuesday, 28-Oct-2003 14:17:31 CET
Parent Server Generation: 0
Server uptime: 2 minutes 51 seconds
Total accesses: 11 - Total Traffic: 36 kB
CPU Usage: u.01 s.01 cu0 cs0 - .0117% CPU load
.0643 requests/sec - 215 B/second - 3351 B/request
1 requests currently being processed, 7 idle workers

______W_........................................................
................................................................
................................................................
................................................................

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process
```

Si provocamos una gran carga en el servidor (podemos usar para ello algunos programas especializados o el propio `ab` proporcionado con Apache), veremos cómo el resultado de `mod_status` se complica:

Figura 43.

```
Current Time: Monday, 29-Dec-2003 18:30:12 CET
Restart Time: Sunday, 28-Dec-2003 04:02:17 CET
Parent Server Generation: 2
Server uptime: 1 day 14 hours 27 minutes 55 seconds
Total accesses: 82251 - Total Traffic: 12.7 GB
CPU Usage: u12.04 s6.11 cu0 cs0 - .0131% CPU load
.594 requests/sec - 96.4 kB/second - 162.3 kB/request
101 requests currently being processed, 9 idle workers

WWW__WWWWWWWWWWWWWWWWWWWWW_CC..WWWWWWWWWWWWWW_WWWWWWWWW_WWWW_WWW
WWWWWWWWWWWWWWWWWWWWWWWWW__WWW_WWWWWWWWWWWWWWWW.................
................................................................
................................................................

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process
```

En esta información que nos proporciona `mod_status`, podemos observar lo que Apache denomina el tablero (*scoreboard*) donde se representan todos los *slots* o procesadores de peticiones y el estado de éstos. De la información que tenemos en pantalla podemos deducir que muchos de estos *slots* están ocupados enviando información (los marcados con W). Esto, en principio, indica que nuestro servidor está respondiendo bien a las peticiones recibidas. Cabe indicar aquí que esta prueba se ha realizado enviando un total de cien peticiones simultáneas al servidor hasta completar 100.000 peticiones, siendo la máquina cliente de estas peticiones el propio servidor. Vemos también en la información mostrada que el servidor aún dispone de algunos *slots* libres. Esto nos indicaría que todavía estamos en disposición de recibir más peticiones (no nos indica que seamos capaces de atenderlas con la velocidad deseada).

9.3.2. Obtener información de rendimiento del sistema

Otra fuente de información sobre cómo se está comportando nuestro servidor es el sistema operativo. Por norma general, los sistemas operativos suelen incluir una serie de herramientas muy variadas que nos permiten saber en todo momento el estado de éste. Dichas herramientas nos permitirán conocer el nivel de uso de procesador, de memoria, etc.

Usaremos para todo el análisis la serie de herramientas que suelen estar disponibles de forma habitual en los sistemas Unix. En caso de

ANOTACIONES

que nuestro sistema no disponga de dichas herramientas, muchas de ellas están disponibles libremente en Internet. También existen herramientas similares para la mayoría de sistemas operativos.

Carga de procesador

Mediante este término nos referimos al nivel de ocupación de las unidades centrales de proceso, CPU, del sistema servidor. El funcionamiento normal de los servidores web, como Apache, requiere de un cierto nivel de uso de procesador, nivel que se ve incrementado por el uso de páginas generadas dinámicamente, etc.

Si, como ocurre en algunos casos, el sistema del servidor web proporciona más servicios que los del web (el servidor de correo, por ejemplo) deberemos ser muy cuidadosos con el nivel de procesador utilizado.

Podemos disponer de una primera aproximación al uso de nuestro sistema con el comando: `vmstat`.

```
[carlesm@bofh carlesm]$ vmstat 2
 procs                                 memory       swap      io        system          cpu
 r   b   w    swpd    free    buff    cache   si    so   bi  bo   in    cs    us   sy  id
 1   0   0   26364   46412  178360   206072   0     0    4  13   29    41     7    0  46
 0   0   0   26364   46412  178360   206072   0     0    0   0  107    22     0    0 100
 0   0   0   26364   46412  178360   206072   0     0    0   0  108    22     0    0 100
```

En este ejemplo, observamos la distribución de carga de CPU. Son los tres valores situados a la derecha, etiquetados con `us`, `sy` e `id`, que corresponden a: *user*, *system* y *idle* (respectivamente, usuario, sistema y desocupada). Estos valores nos marcan qué porcentaje del tiempo permanece el procesador en cada uno de estos estados:

- *user*: el procesador permanece en espacio de usuario mientras ejecuta programas.

- *system*: el procesador permanece en este estado mientras ejecuta código que forma parte del núcleo del sistema operativo, o atiende a llamadas al sistema, como pueden ser controladores de comunicaciones, etc.

ANOTACIONES

- *idle*: es el tiempo que el procesador permanece libre, no ocupado.

Un valor alto continuado de `us` indicaría un uso intenso de procesador. En ese caso, la máquina se encuentra cerca de su límite de respuesta y deberemos estudiar soluciones para evitar que un aumento de carga implique una pérdida de capacidad de respuesta. Dichas soluciones deben orientarse a conseguir un menor consumo de procesador (reescritura de código, optimización de éste) o una ampliación de la capacidad de proceso.

Un valor alto de `sy` indica que el sistema permanece mucho tiempo ocupado con tareas del propio núcleo del sistema. Debemos intentar averiguar las causas (controladores incorrectos o defectuosos, *hardware* no apropiado, etc.) y solucionarlas.

Un valor alto de `id` (si tenemos problemas de rendimiento) indicaría que el problema no se encuentra en el procesador, debiendo dirigir nuestras pesquisas hacia otros aspectos.

Otros valores importantes que nos muestra `vmstat` son las dos columnas anteriores:

- `in` (*interrupts*): el número de interrupciones que han ocurrido por segundo (incluyendo las correspondientes al reloj del sistema).

- `Cs` (*context switches*): el número de cambios de contexto (de proceso o hebra activa en el procesador) que han ocurrido por segundo.

Un valor excesivamente alto de `cs` suele indicar que el número de procesos del sistema es excesivo. Si dicho exceso viene provocado por los procesos que genera el servidor web, deberemos ajustar ese número a la baja. Otra posible causa derivaría de un nivel de comunicaciones inter-proceso (IPC) elevada y poco óptima que provocase cambios de contexto excesivos.

Las tres primeras columnas nos indican otros valores que conviene observar:

- `r`: número de procesos listos para ejecutar.

ANOTACIONES

- b: número de procesos bloqueados.

- w: número de procesos pasados a memoria *swap* pero ejecutables.

En situaciones de carga, dichos indicadores de número de procesos nos pueden dar una idea del nivel de contención para entrar en el procesador que hay que ejecutar:

23	0	1	27328	7944	169696	210932	0	0	0	480	116	13715	41	59	0
21	0	0	27336	7576	169624	211376	0	4	0	4	104	13724	43	57	0
17	0	1	27336	7096	169448	211924	0	0	0	474	113	13726	40	60	0
13	0	0	27344	6624	169444	212296	0	4	0	4	105	13753	38	62	0

Podemos ver en el ejemplo, obtenido en un momento de carga elevada del servidor, que tenemos un elevado número de procesos disponibles para ejecutar y un nivel de desocupación de procesador 0. Como estos datos proceden de una máquina monoprocesadora y viendo, además, el número de cambios de contexto que se efectúan, podemos deducir que el número de procesos que está ejecutándose es excesivo.

Existen otras herramientas, top, ps, entre otras, que nos proporcionarán la misma información o información complementaria. Es realmente importante que conozcamos las herramientas de que dispongamos en nuestro sistema operativo, así como las capacidades de éstas.

Ocupación de memoria

El mismo comando, vmstat, nos muestra además algunos datos básicos sobre la ocupación de memoria. Disponemos de las siguientes columnas con información sobre ocupación de memoria:

Tenemos las columnas swpd, free, buff y cache que nos indican, respectivamente:

- swpd: la ocupación de la memoria *swap* en disco (memoria de intercambio).

ANOTACIONES

- `free`: la cantidad de memoria física (RAM) libre.
- `buff`: la cantidad de memoria usada como *buffers*.
- `cache`: la cantidad de memoria usada como *cache* de disco.

Durante el tiempo en que nuestro servidor web esté bajo un nivel de carga elevado, debemos observar atentamente dichos valores. Un nivel muy alto de `swpd` y uno muy bajo de `free`, `buff` y `cache` indicaría que nuestro sistema no dispone de memoria suficiente y que debe recurrir a la memoria virtual en disco, mucho más lenta que la memoria RAM.

Otras dos columnas, `si` y `so`, nos indican la cantidad de memoria que se envía a la memoria virtual en disco o la que se recupera de allí en kB/s. Valores diferentes de cero y sostenidos en el tiempo indicarían que nuestro sistema está falto de memoria y, por lo tanto, debe descartar y recuperar de disco datos continuamente.

Acceso a disco

Uno de los puntos que suele descuidarse al dimensionar equipamiento para los servidores web es el del acceso a disco. Debemos tener en cuenta que un servidor web envía constantemente datos que lee del disco a los clientes remotos (las páginas, imágenes, etc.). Por eso, unos tiempos de acceso a disco pequeños y unas velocidades de transferencia elevadas posibilitarán que el servidor web tenga un alto nivel de rendimiento al servir páginas.

Para tener una idea de cómo están respondiendo nuestros discos a las peticiones, podemos usar, por un lado, el propio comando `vmstat` y, por el otro, un comando más especializado: `iostat`. El resultado de ejecutar `iostat` será:

```
avg-cpu:   %user   %nice           %sys          %idle
           67.50   0.00            18.50         14.00

Device:            tps             Blk_read/s    Blk_wrtn/s    Blk_read     Blk_wrtn
dev3-0             32.00           208.00        844.00        416          1688
```

Nota

El uso de disco puede venir tanto por acceso a datos por parte de programas como por uso de *swap*. En el segundo caso, la mejor...

En este resultado podemos ver la cantidad de transacciones de acceso a disco que se han producido, el número de bloques (sectores) leídos y escritos y el total de bloques durante el tiempo de medida.

Valores muy altos indicarían un uso elevado del disco. Debemos entonces asegurarnos de que el sistema dispone de discos rápidos, con el menor tiempo de acceso posible, la mayor velocidad de transferencia disponible y de suficiente memoria para que pueda realizar *cache* de disco de forma eficiente o para evitar un uso excesivo de *swap*.

9.3.3. Mejoras en la configuración

Hay algunas mejoras que podemos conseguir mediante ajustes a la configuración del servidor web. Diferentes versiones de Apache incorporan dichos ajustes por defecto, pero aún así deberemos asegurar los valores que esté usando el servidor web, ya que un cambio de éstos puede tener efectos drásticos en el rendimiento del sistema.

Consultas al DNS

Uno de los puntos que suele representar un cuello de botella al procesar peticiones es el hecho de que, en determinadas circunstancias, Apache envía consultas al DNS por cada acceso. Por defecto, dicho comportamiento está inhabilitado desde la versión 2.0. No obstante, existe un caso en el que aún se deben realizar consultas al DNS por cada petición recibida: se trata de cuando usamos directivas de control de acceso, como `Allow`. En este caso, es recomendable, siempre que sea posible, usar direcciones IP en lugar de nombres.

Enlaces simbólicos y Overrides

Si usamos las opciones `FollowSymLinks` o `SymLinksIfOwnerMatch`, Apache deberá comprobar para cada petición si ésta se trata de un enlace, así como si alguno de los directorios padre en la jerarquía de directorios es un enlace simbólico. Ello supone una penalización importante de tiempo por cada acceso. Por eso deberíamos desactivar dichas opciones siempre que nos sea posible, o en caso de requerirlas en algún espacio de disco concreto,

ANOTACIONES

limitar su ámbito mediante las directivas de configuración de Apache (`Directory`, etc.).

Igualmente si usamos directivas de tipo `AllowOverride`, esto significará que por cada acceso a un fichero, Apache buscará un fichero `.htaccess` en la jerarquía de directorios que preceda a dicho fichero. Debemos, al igual que en el caso anterior, limitar el ámbito de aplicación de dicha directiva a lo estrictamente necesario.

Mapeo de memoria y `sendfile`

Debemos, siempre que nuestra plataforma lo permita, asegurarnos que Apache utiliza, para acceder al contenido de un fichero, las capacidades del sistema operativo para mapeo de memoria (`mmap`). Por norma general, dicho uso significará un aumento de rendimiento importante. Debemos, eso sí, consultar la documentación de Apache para nuestra plataforma, pues existen algunos sistemas operativos en los que el uso de`mmap` supone una pérdida de rendimiento. También debemos considerar que los ficheros accesibles a través de unidades compartidas de red (NFS por ejemplo), no deben ser mapeados a memoria.

Otra capacidad del sistema operativo que supone un incremento sustancial del rendimiento de Apache es el uso de la llamada de sistema `sendfile`, que es una función que proporcionan algunos sistemas operativos, la cual se caracteriza por delegar en el núcleo del sistema operativo la tarea de enviar un fichero a través de la red. En caso de disponer de dicha directiva, conviene asegurarse de que Apache, en tiempo de compilación, la utiliza. Debemos, eso sí, tener las mismas precauciones que con `mmap`, es decir, asegurarnos de que nuestra plataforma está soportada y de que no se trata de ficheros accesibles a través de unidades de disco de red.

Creación de procesos y hebras

Otro de los puntos donde podemos controlar el rendimiento de Apache es en la creación e instanciación de procesos. Al arrancar, Apache crea una serie de procesos que atenderán las peticiones. Cuando un proceso ha atendido un número determinado de peticio-

ANOTACIONES

nes, se finaliza y otro se arranca en su lugar. Podemos ajustar dicho comportamiento mediante:

- `MinSpareServers`: número de procesos servidores mínimo que debemos tener en ejecución.
- `MaxSpareServers`: número de procesos servidores máximo sin atender ninguna petición que podremos tener en ejecución.
- `StartServers`: número de procesos servidores que podemos arrancar.
- `MaxRequestsPerChild`: peticiones máximas que podrá atender un proceso antes de ser reciclado.

Otro aspecto que podemos usar para controlar el funcionamiento de Apache es el del gestor de procesamiento (MPM). Por defecto, Apache funciona con un gestor de procesamiento (MPM) basado en procesos de sistema, llamado `prefork`, pero podemos cambiar éste por uno llamado `worker`, que además, para cada proceso de sistema lance una serie de hebras *(threads*). Este último es una buena elección en aquellos sistemas con un nivel de carga elevado.

ANOTACIONES

Bibliografía

Apache Foundation (2003). *Apache HTTP Server Version 2.0 Documentation.* http://httpd.apache.org/docs-2.0/: Apache Foundation.

Atkinson, L.; Suraski, Z. (2003). *Core PHP Programming, Third Edition.* Prentice Hall.

Bequet, H. (2001). *Professional Java SOAP.* Wrox Press.

Bergsten, H. (2002). *Java Server Pages.* O'Reilly.

Bowen, R.; Lopez Ridruejo, D.; Liska, A. (2002). *Apache Administrator's Handbook.* SAMS.

Chappell, D.A.; Jewell, T. (2002). *Java Web Services.* O'Reilly.

Flanagan, D. (2001). *JavaScript: The Definitive Guide.* O'Reilly.

Flanagan, D. (2002). *Java in a Nutshell, 4th Edition.* O'Reilly.

Flanagan, D.; Farley, J.; Crawford, W.; Magnusson, K. (1999). *Java Enterprise in a Nutshell.* O'Reilly.

Goodman, D. (1998). *Dynamic HTML. The definitive reference.* O'Reilly.

Goodman, D. (2002). *Dynamic HTML: The Definitive Reference* .O'Reilly.

Gundavaram, S. (1996). *CGI programming.* O'Reilly.

Hamilton, G.; Cattell, R.; Fischer, M. (1998). *JDBC Database Access with Java.* Addison-Wesley.

Hunter, J. (2001). *Java Servlet Programming.* O'Reilly.

Laurie, B.; Laurie, P. (2002). *Apache: The Definitive Guide, 3rd Edition* . O'Reilly.

Meyer, Eric A. (2000). *Cascading Style Sheets: The Definitive Guide.* O'Reilly.

ANOTACIONES

Musciano, C.; Kennedy, B. (2000). *HTML & XHTML: The Definitive Guide.* O'Reilly.

Raggett, D.; Lam, J.; Alexander,I.; Kmiec, M. (1998). *Raggett on HTML 4.* Addison Wesley Longman Limited. Capítulo 2 disponible en línea en: http://www.w3.org/People/Raggett/book4/ch02.html.

Ray, E.T. (2001). *Learning XML.* O'Reilly.

Redding, Loren E. (2001). *Linux Complete.* Sybex.

Reese, G. (2000). *Database Programming with JDBC and Java.* O'Reilly.

Rosenfeld, L.; Morville, P. (1998). *Information Architecture for the World Wide Web.* O'Reilly.

Rusty Harold,E.; Means, W.S. (2002). *XML in a Nutshell, 2 nd Edition.* O'Reilly.

Tidwell, D. (2001). *XSLT.* O'Reilly.

van der Vlist, E. (2001). *XML Schema.* O'Reilly.

Wainwright, P. (2002). *Professional Apache 2.0.* Wrox Press.

World Wide Web (W3) Consortium (2003). *http://www.w3.org/Consortium/* World Wide Web Consortium.

ANOTACIONES

GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

ANOTACIONES

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

ANOTACIONES

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent.

An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

ANOTACIONES

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition.

Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

ANOTACIONES

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material.

If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

ANOTACIONES

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

ANOTACIONES

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

ANOTACIONES

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number.

Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

ANOTACIONES

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit.

When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form.

Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the

ANOTACIONES

translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

ANOTACIONES